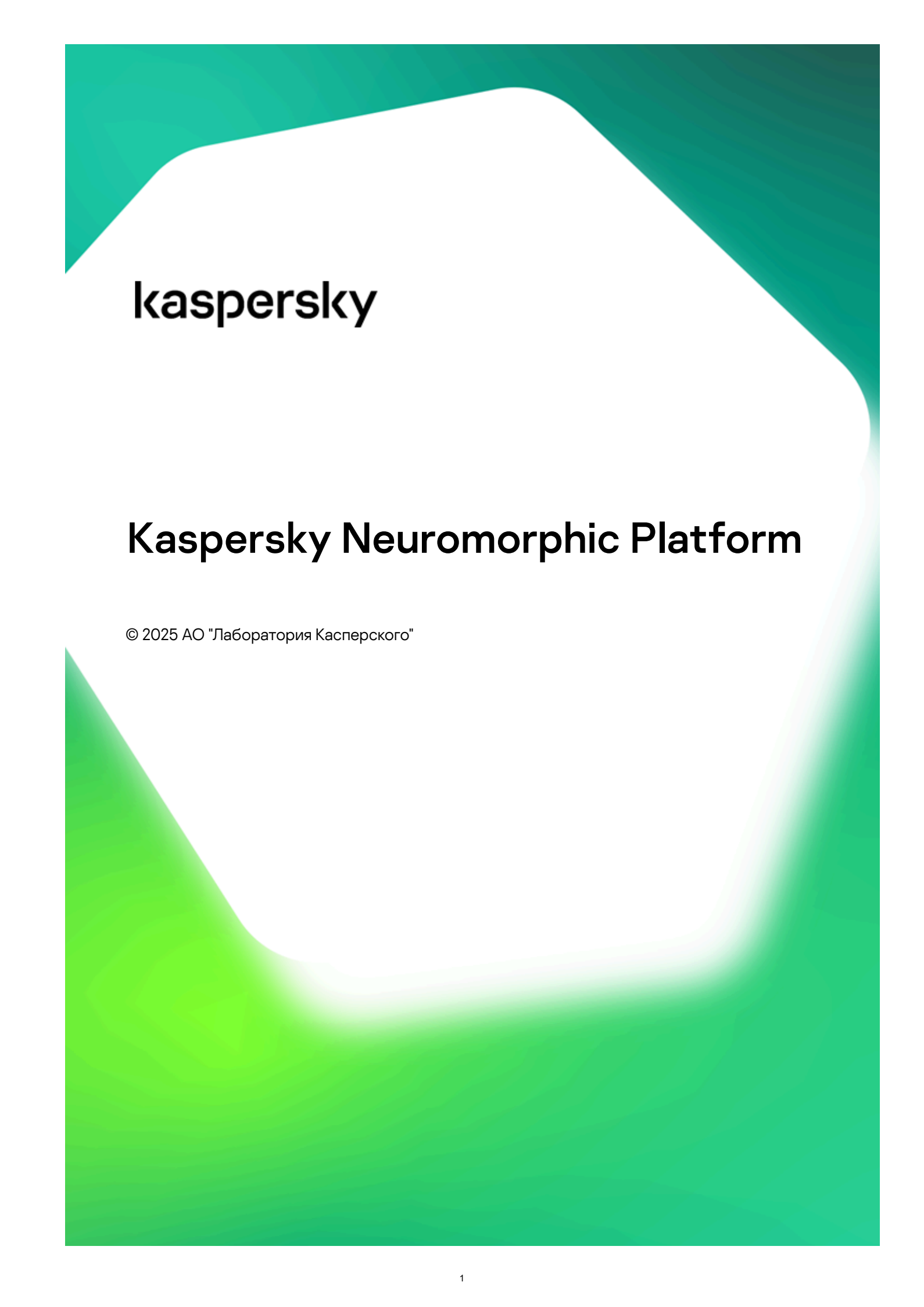


The background of the slide is a vibrant green with a white, irregular, rounded shape in the center. The word "kaspersky" is written in a bold, black, lowercase sans-serif font in the upper left area of the white shape. Below it, the title "Kaspersky Neuromorphic Platform" is written in a bold, black, uppercase sans-serif font. At the bottom left of the white shape, there is a small copyright notice. The overall design is clean and modern.

kaspersky

Kaspersky Neuromorphic Platform

© 2025 АО "Лаборатория Касперского"

Содержание

[О Kaspersky Neuromorphic Platform](#)

[Комплект поставки](#)

[Комплект поставки для разработки прикладных решений на Linux](#)

[Комплект поставки для разработки прикладных решений на языке программирования Python на Linux](#)

[Комплект поставки для работы с исходным кодом платформы](#)

[Аппаратные и программные требования](#)

[Что нового](#)

[Основные понятия Kaspersky Neuromorphic Platform](#)

[Нейроны импульсной нейронной сети](#)

[Нейрон по модели AltAILIF](#)

[Нейрон по модели BLIFATNeuron](#)

[Нейрон по модели SynapticResourceSTDPBLIFATNeuron](#)

[Синапсы импульсной нейронной сети](#)

[Синапс по модели AdditiveSTDPDeltaSynapse](#)

[Синапс по модели DeltaSynapse](#)

[Синапс по модели SynapticResourceSTDPDeltaSynapse](#)

[Нейроморфный процессор Алтай-2](#)

[Бэкенды Kaspersky Neuromorphic Platform](#)

[Бэкенды для исполнения импульсных нейронных сетей](#)

[Бэкенды для обучения и исполнения искусственных нейронных сетей](#)

[Формат описания нейронной сети](#)

[Форматы данных, обрабатываемые платформой](#)

[Архитектура Kaspersky Neuromorphic Platform](#)

[Изменение параметров конфигурации программного эмулятора](#)

[Общие принципы работы Kaspersky Neuromorphic Platform](#)

[Обмен данными в Kaspersky Neuromorphic Platform](#)

[Работа бэкенда для CPU](#)

[Обработка сообщений по подписке](#)

[Установка и удаление платформы](#)

[Установка deb-пакетов](#)

[Установка пакетов для разработки на языке программирования Python](#)

[Удаление платформы](#)

[Сценарий сборки проекта платформы или прикладного решения на C++](#)

[Загрузка и распаковка архива с исходным кодом платформы](#)

[Определение параметра Boost_ROOT в ОС Windows](#)

[Особенности сборки платформы с Python-фреймворком](#)

[Конфигурация сборки платформы или прикладного решения в ОС Windows](#)

[Настройка параметров сборки проекта прикладного решения на C++](#)

[Компоненты платформы для использования на C++](#)

[Библиотека neuron_traits](#)

[Библиотека synapse_traits](#)

[Библиотека core](#)

[Пространство имен messaging](#)

[Пространство имен tags](#)

[Класс Backend](#)

[Структура BaseData](#)

[Класс continuously_uid_generator](#)

[Класс Device](#)

[Класс MessageBus](#)

[Класс MessageEndpoint](#)

[Класс Population](#)

[Класс Projection](#)

[Класс Subscription](#)

[Класс TagMap](#)

[Класс UID](#)

[Функция uid_hash\(\)](#)

[Библиотека meta](#)

[Пространство имен devices](#)

[Фреймворк для C++](#)

[Пространство имен coordinates](#)

[Пространство имен data_processing](#)

[Пространство имен inference_evaluation](#)

[Пространство имен input](#)

[Пространство имен modifier](#)

[Пространство имен monitoring](#)

[Пространство имен normalization](#)

[Пространство имен output](#)

[Пространство имен population](#)

[Пространство имен projection](#)

[Пространство имен sonata](#)

[Пространство имен storage](#)

[Пространство имен synchronization](#)

[Класс BackendLoader](#)

[Класс Model](#)

[Класс ModelExecutor](#)

[Класс ModelLoader](#)

[Класс Network](#)

[Решение типовых задач на C++](#)

[Добавление нового типа нейрона](#)

[Добавление нового типа синапса](#)

[Подготовка модели и ее исполнение на бэкенде](#)

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

[Компоненты платформы для использования на Python](#)

[Модуль neuron_traits](#)

[Модуль synapse_traits](#)

[Модуль core](#)

[Модуль messaging](#)

[Класс Backend](#)

[Структура BaseData](#)

[Класс continuously_uid_generator](#)

[Класс Device](#)

[Класс MessageBus](#)

[Класс MessageEndpoint](#)

[Набор классов Population](#)

[Набор классов Projection](#)

[Набор классов Subscription](#)

[Класс TagMap](#)

[Класс UID](#)

[Класс uid_hash](#)

[Класс UUID](#)

[Python-фреймворк](#)

[Пакет ANN2SNN](#)

[Тернарные слои пакета ANN2SNN](#)

[Модуль input](#)

[Модуль output](#)

[Модуль sonata](#)

[Класс BackendLoader](#)

[Класс ImpactMessageObserver](#)

[Класс Model](#)

[Класс ModelExecutor](#)

[Класс ModelLoader](#)

[Класс Network](#)

[Класс SpikeMessageObserver](#)

[Решение типовых задач на Python](#)

[Исполнение нейронной сети, загруженной на бэкенд вручную](#)

[Сценарий исполнения нейронной сети из тернарных слоев для классификации изображений](#)

[Создание и обучение нейронной сети из тернарных слоев для классификации изображений](#)

[Запуск инференса нейронной сети из тернарных слоев для классификации изображений](#)

[Сценарий исполнения регрессионной модели с обучаемыми блоками, кодирующие и декодирующие данные](#)

[Создание и обучение регрессионной модели](#)

[Запуск инференса регрессионной модели](#)

[Использование Kaspersky Neuromorphic Platform API](#)

[Участие в разработке платформы](#)

[Лицензирование платформы](#)

[О предоставлении данных](#)

[Устранение неисправностей](#)

[Не выполняется сборка платформы с Python-фреймворком](#)

[Компилятор или система сборки не может найти библиотеку Boost в ОС Windows](#)

[Приложения](#)

[Структура файла, описывающего группы нейронов](#)

[Структура файла, описывающего группы синапсов](#)

[Структура и пример файла базового формата](#)

[Результаты оценки качества модели, передаваемые в формате CSV](#)

[Формулы для расчета метрик оценки качества модели](#)

[Параметры и пример YAML-файла, содержащего конфигурацию нейроморфного процессора Алтай-2](#)

[Параметры конфигурации сборки, специфичные для Kaspersky Neuromorphic Platform](#)

[Пресеты для сборки платформы](#)

Глоссарий

[Artificial Neural Network \(ANN\)](#)

[Spiking Neural Network \(SNN\)](#)

[Бэкенд](#)

[Бэкенд AltAI ANN2SNN](#)

[Бэкенд AltAI SNN](#)

[Вес синапса](#)

[Внешняя среда](#)

[Вычислитель](#)

[Механизм Winner Takes ALL \(WTA\)](#)

[Модификатор](#)

[Нейрон](#)

[Период ППС](#)

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

[Популяция](#)

[Проекция](#)

[Синапс](#)

[Синаптическое воздействие](#)

[Соединение](#)

[Спайк](#)

[Такт](#)

[Тик](#)

[Фреймворк](#)

[Функция нейрона](#)

[Функция синапса](#)

[Информация о стороннем коде](#)

[Уведомления о товарных знаках](#)

О Kaspersky Neuromorphic Platform

Kaspersky Neuromorphic Platform (далее также KNP, платформа) – это программная платформа, предназначенная для разработки импульсных нейронных сетей, их обучения и исполнения на разных вычислителях.

С помощью Kaspersky Neuromorphic Platform вы можете:

- Создавать [импульсные нейронные сети](#) (англ. Spiking Neural Network, SNN) и обучать их на различных типах входных данных (например, телеметрии, событиях, изображениях, 3D-данных, звуковых и тактильных данных).
- Преобразовывать [искусственные нейронные сети](#) (англ. Artificial Neural Network, ANN) в импульсные и обучать их.
- Проводить прикладные исследования в области классификации входных данных и других областях применения 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 в области робототехнических манипуляторов, интернета вещей, беспилотных систем, человеко-машинного взаимодействия, носимых устройств и оптимизационного планирования.
- Реализовывать прикладные решения на устройствах с пониженным энергопотреблением, использующих нейроморфные процессоры (англ. neuromorphic processors).

Для решения вышеперечисленных задач вы можете использовать языки C++ и Python®. Платформа поддерживает CPU, GPU с поддержкой технологии CUDA, а также нейроморфный процессор Алтай-2, предназначенный для энергоэффективного исполнения нейронных сетей в разных классах интеллектуальных устройств.

Комплект поставки

Этот раздел содержит информацию о комплектах поставки Kaspersky Neuromorphic Platform. Платформа имеет несколько вариантов комплекта поставки. Вы можете выбрать тот вариант, который наиболее подходит для ваших целей использования платформы.

Для сборки платформы или прикладного решения также поставляется Docker-образ `knp-build-image`. Docker-образ необходимо скачать с [репозитория Docker Hub](#).

Комплект поставки для разработки прикладных решений на Linux

Комплект поставки может быть использован для разработки прикладных решений в операционной системе Linux.

В комплект поставки входят следующие файлы:

- Архив `pkg-deb.zip` со следующими пакетами:
 - `knp-cpu-single-threaded-backend_<knp_version>_amd64.deb` – deb-пакет с динамической библиотекой однопоточного бэкенда для CPU.
 - `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb` – deb-пакет с динамической библиотекой многопоточного бэкенда для CPU.
 - `knp-gpu-cuda-backend_<knp_version>_amd64.deb` – deb-пакет с библиотекой бэкенда для GPU с поддержкой технологии CUDA.
 - `knp-cpp-framework_<knp_version>_amd64.deb` – deb-пакет с динамической библиотекой фреймворка для C++, содержащий:
 - Бинарный код библиотек платформы.
 - Файл с условиями использования платформы `LICENSE.txt` на английском языке.
 - Файл с информацией о платформе `README.md` на английском языке.
 - dev-deb пакеты для каждого deb-пакета:
 - `knp-cpu-single-threaded-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb`.
 - `knp-cpu-multi-threaded-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb`.
 - `knp-gpu-cuda-backend-dev_<knp_version>_amd64.deb` – dev-deb-пакет для deb-пакета `knp-gpu-cuda-backend_<knp_version>_amd64.deb`.
 - `knp-cpp-framework-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpp-framework_<knp_version>_amd64.deb`.
- В каждый dev-deb пакет входят следующие файлы:
- Заголовочные файлы.
 - Модули CMake для сборки платформы на C++.
 - Статические версии библиотек платформы.
- `knp-python3-framework_<knp_version>_amd64.deb` – deb-пакет с Python-фреймворком для работы на бэкенде AltAI SNN.
 - `knp-examples_<knp_version>_amd64.deb` – deb-пакет с примерами использования Kaspersky Neuromorphic Platform.
 - `knp-documentation_<knp_version>_all.deb` – deb-пакет со справочником API.
 - `knp-unspecified_<knp_version>_amd64.deb` – deb-пакет со служебными файлами CMake.
- Пакеты, предоставляемые компанией ООО "Мотив HT":

- `knp-altai-backend_<motiv_version>_amd64.deb` – deb-пакет с динамической библиотекой [бэкенда AltAI SNN](#).
- `knp-altai-backend-dev_<motiv_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-altai-backend_<motiv_version>_amd64.deb`.
- `knp_ann2snn.deb` – deb-пакет с пакетом ANN2SNN, динамической библиотекой [бэкенда AltAI ANN2SNN](#) для [нейроморфного процессора Алтай-2](#) и компонентом для размещения нейронной сети на нейроморфном процессоре Алтай-2.

где:

- `<knp_version>` – версия Kaspersky Neuromorphic Platform;
- `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-2 и пакета ANN2SNN.

Комплект поставки для разработки прикладных решений на языке программирования Python на Linux

Комплект поставки может быть использован для разработки прикладных решений на языке программирования Python в операционной системе Linux.

В комплект поставки Kaspersky Neuromorphic Platform входят следующие файлы:

- Архив `pkg-deb.zip` с deb-пакетом `knp-python3-framework_<knp_version>_amd64.deb`, содержащим Python-фреймворк для работы на бэкенде AltAI SNN.
- Архив `pkg-python.zip` с whl-пакетом `knp-<knp_version>-py3-none-any.whl`, содержащий Python-фреймворк для работы на бэкенде AltAI SNN.
- Пакеты, предоставляемые ООО "Мотив НТ":
 - `knp_ann2snn.deb` – deb-пакет с пакетом ANN2SNN, динамической библиотекой бэкенда AltAI ANN2SNN для [нейроморфного процессора Алтай-2](#) и компонентом для размещения нейронной сети на нейроморфном процессоре Алтай-2.
 - `knp_ann2snn_x86_64-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом ANN2SNN, динамической библиотекой бэкенда AltAI ANN2SNN для [нейроморфного процессора Алтай-2](#) и компонентом для размещения нейронной сети на нейроморфном процессоре Алтай-2.
Этот пакет можно установить на компьютер с процессором архитектуры x86.
 - `knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом ANN2SNN и динамической библиотекой бэкенда AltAI ANN2SNN для [нейроморфного процессора Алтай-2](#).
Этот пакет можно установить на компьютер с процессором архитектуры ARM.

где:

- `<knp_version>` – версия Kaspersky Neuromorphic Platform;
- `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-2 и пакета ANN2SNN.

Комплект поставки для работы с исходным кодом платформы

Комплект поставки может быть использован для разработки прикладных решений на C++ на Linux и в ОС Windows.

Вы можете получить файлы с исходным кодом Kaspersky Neuromorphic Platform путем клонирования [репозитория платформы](#). Вы также можете получить файлы с исходным кодом в виде архива.

Комплект поставки включает следующие файлы:

- Исходный код платформы.
- Справочник API.
- Файл с информацией о платформе `README.md` на английском языке.
- Файл с условиями использования платформы `LICENSE.txt` на английском языке.
- Файл с информацией о стороннем коде `NOTICE.txt` на английском языке.
- Файл с правилами участников репозитория `CONTRIBUTING.md` на английском языке.
- Файл с условиями участия в репозитории `CLA.md` на английском языке.

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;
- графический процессор с поддержкой технологии CUDA: процессор NVIDIA с архитектурой Pascal и выше;
- нейроморфный процессор (при использовании бэкендов `AltAI_SNN` или `AltAI_ANN2SNN`): Алтай-2;
- 8 ГБ оперативной памяти;
- Свободного пространства на жестком диске:
 - 1 ГБ для установки Kaspersky Neuromorphic Platform.
 - 10 ГБ для сборки платформы или прикладного решения.

Поддерживаемые операционные системы:

- Debian GNU/Linux 12.5 и выше;
- Ubuntu 22.04 LTS и выше;
- Windows 7;
- Windows 10.

Вы можете использовать устройство под управлением любой другой операционной системы из семейства Linux, если дистрибутив операционной системы содержит библиотеку Boost версии 1.81 и выше.

Для работы с платформой на устройстве должно быть установлено следующее программное обеспечение:

- Visual Studio 2022 и выше.

Это программное обеспечение требуется установить для [сборки платформы или прикладного решения](#) в ОС Windows.

- Boost 1.81 и выше

При работе в ОС Windows рекомендуется установить библиотеку, [скомпилированную](#)  предварительно для Visual Studio. Например, для Visual Studio 2022 версия компилятора – 14.3.

- CMake 3.25 и выше.

- CUDA Toolkit 12.8.

Это программное обеспечение требуется установить для работы на компьютере с графическим процессором NVIDIA.

- ZeroMQ 4.3.5.

Это программное обеспечение требуется установить для работы на компьютере с центральным процессором или графическим процессором NVIDIA.

- При установке deb-пакетов с компонентами платформы для работы на [бэкенде AltAI ANN2SNN](#)  должно быть установлено следующее программное обеспечение:

- TensorFlow 2.16.2 и выше.

- Keras 3.12 и выше.

- NumPy 2.16.2 и выше.

Версия библиотеки NumPy должна соответствовать версии библиотеки TensorFlow.

- При установке whl-пакета или deb-пакета с Python-фреймворком для работы на [бэкенде AltAI SNN](#)  на устройстве должен быть установлен Python 3.12 и выше.

- При установке whl-пакета с Python-фреймворком для работы на бэкенде **AltAI ANN2SNN** на устройстве должно быть установлено следующее программное обеспечение:
 - NumPy 1.24.3.
 - TensorFlow 2.16.2.
 - Loguru 0.7.2.
 - PyYAML 6.0.1.
 - NetworkX 3.1.
 - Matplotlib 3.6.3.
 - tqdm 4.66.5.
- При установке deb-пакета с Python-фреймворком для работы на бэкенде **AltAI ANN2SNN** на устройстве должно быть установлено следующее программное обеспечение:
 - python3-pip.
 - python3-numpy.
 - python3-yaml.
 - python3-networkx.
 - python3-matplotlib.
 - python3-loguru.
 - python3-tqdm.
 - TensorFlow 2.16.2.

Kaspersky Neuromorphic Platform использует следующие сторонние библиотеки:

Boost

Библиотеки используются для реализации классов и структур (например, идентификатора **UID** и загрузчика бэкенда **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.

Что нового

В Kaspersky Neuromorphic Platform 2.0 появились следующие возможности и доработки:

- [Комплект поставки для разработки прикладных решений на Linux](#) – добавлены пакеты с библиотекой бэкенда для GPU с технологией поддержки CUDA.
- Бэкенды:
 - Бэкенд [CUDABackend](#) – добавлен интерфейс для GPU-бэкенда.
 - Бэкенд [SingleThreadedCPUBackend](#) – добавлена поддержка нейрона по модели [AltAILIF](#).
 - Бэкенд [MultiThreadedCPUBackend](#) – добавлена поддержка нейрона по модели [SynapticResourceSTDPBLIFATNeuron](#) и синапса по модели [SynapticResourceSTDPDeltaSynapse](#).
 - Оптимизирована работа CPU-бэкенда.
- [Компоненты для использования на C++](#):
 - Библиотека [core](#) – расширена функциональность для классов [Backend](#), [MessageBus](#), [MessageEndpoint](#), [Population](#), [Projection](#), [Subscription](#) и [TagMap](#).
 - [Фреймворк для C++](#) – расширена функциональность фреймворка для C++:
 - Пространство имен [data_processing](#) – добавлено пространство имен, предоставляющее функциональность для обработки данных для обучения и тестирования моделей классификации.
 - Пространство имен [inference_evaluation](#) – добавлено пространство имен, предоставляющее функциональность для оценки качества модели.
 - Пространство имен [modifier](#) – добавлено пространство имен, предоставляющее обработчики сообщений.
 - Пространство имен [normalization](#) – добавлено пространство имен, предоставляющее функции и функторы для нормализации параметров нейронов и синапсов в нейронных сетях.
 - Пространство имен [monitoring](#) – добавлено пространство имен [model](#) с функциями для мониторинга работы модели.
 - Пространство имен [projection](#) – добавлены функции для генерации синапсов и проекций синапсов.
 - Класс [ModelExecutor](#) – добавлены методы.
 - Библиотека [neuron_traits](#) – расширены интерфейсы нейронов по моделям [BLIFATNeuron](#) и [SynapticResourceSTDPBLIFATNeuron](#).
- [Компоненты для использования на Python](#) – добавлены методы для класса [ModelExecutor](#) и [UID](#).

- Библиотека `synapse_traits` (для использования на C++ и Python) – расширен интерфейс синапса по модели `SynapticResourceSTDPDeltaSynapse`.
- Пример [mnist-learn](#) – добавлен пример обучения нейронной сети на изображениях рукописных чисел и их меток из базы данных MNIST и ее исполнения.

Основные понятия Kaspersky Neuromorphic Platform

Этот раздел содержит развернутые определения основных понятий в Kaspersky Neuromorphic Platform.

Нейроны импульсной нейронной сети

Элементарной единицей импульсной нейронной сети является *нейрон*, у которого есть набор входных сигнальных линий (дендриты) и выходная сигнальная линия (аксон). Состояние нейрона описывается нейронным потенциалом. Если в процессе исполнения импульсной нейронной сети потенциал нейрона превышает заданный порог, то нейрон испускает по своему аксону спайк и отправляет его в заданные целевые нейроны. При моделировании некоторых нейронов может применяться механизм утечки, предназначенный для релаксации нейронного потенциала.

В Kaspersky Neuromorphic Platform поддерживаются наборы свойств для следующих нейронов:

- Нейрон по модели [AltAILIF](#).
- Нейрон по модели [BLIFATNeuron](#) (далее также "BLIFAT-нейрон").
- Нейрон по модели [SynapticResourceSTDPBLIFATNeuron](#).

Нейрон по модели AltAILIF

Нейрон по модели AltAILIF представляет собой нейрон-интегратор с утечкой (LIF-нейрон). Нейрон характеризуется потенциалом мембраны и пороговыми значениями потенциала мембраны.

Значение потенциала мембраны изменяется в соответствии со значением [синаптического воздействия](#) ^[2] поступающего от связанного [синапса](#) ^[2]. Если потенциал мембраны нейрона достигает положительного или отрицательного порогового значения, нейрон генерирует положительный или отрицательный спайк соответственно. В зависимости от заданных атрибутов нейрона после превышения одного из порогов потенциал мембраны либо принимает значение величины, на которую был превышен порог, либо сбрасывается до заданного значения.

Значение потенциала мембраны также изменяется линейно на значение утечки, которое в свою очередь может быть как положительным, так и отрицательным.

В атрибутах экземпляра нейрона вы можете указать базовое значение потенциала мембраны, значение потенциала мембраны после сброса, положительное и отрицательное пороговое значение потенциала, значение утечки, а также задать флаги, управляющие поведением потенциала после превышения порогов, флаг сброса потенциала нейрона в начале каждого временного шага и флаг, управляющий поведением утечки в зависимости от текущего знака потенциала нейрона.

Нейрон по модели BLIFATNeuron

Нейрон по модели `BLIFATNeuron` представляет собой нейрон-интегратор с утечкой и адаптивным порогом, генерирующий серию [спайков](#) (англ. Bursting Leaky Integrate-and-Fire with Adaptive Threshold Neuron). 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.

Помимо синаптического воздействия, значение потенциала мембраны может изменяться в силу стохастической стимуляции. При стохастической стимуляции к значению потенциала нейрона каждый такт добавляется случайное число.

Пороговое значение потенциала мембраны нейрона зависит от следующих параметров:

- Базового порогового значения. После генерации спайка потенциал мембраны стремится к базовому значению.
- Динамической составляющей порогового значения. После генерации спайка динамическая составляющая порогового значения 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 значение 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, интервал генерации спайков и параметры стохастической стимуляции в атрибутах экземпляра нейрона.

Нейрон по модели SynapticResourceSTDPBLIFATNeuron

Нейрон по модели `SynapticResourceSTDPBLIFATNeuron` представляет собой BLIFAT-нейрон, который характеризуется свободным синаптическим ресурсом, пороговым значением свободного синаптического ресурса и стабильностью.

Помимо базового порогового значения и динамической составляющей, при расчете порогового значения потенциала мембраны нейрона учитывается дополнительное пороговое значение. Дополнительное пороговое значение в свою очередь зависит от весов и активности синапсов.

Если значение свободного синаптического ресурса по модулю превышает пороговое значение, то этот свободный синаптический ресурс распределяется поровну между всеми пластичными пресинаптическими связями нейрона. После этого значение свободного синаптического ресурса нейрона становится равным 0. Если значение свободного синаптического ресурса по модулю меньше порогового значения, то нейрон присваивает себе синаптический ресурс от связанного синапса до достижения своего порогового значения.

Значение стабильности нейрона изменяется в соответствии с синаптической пластичностью, используемой синапсом, и влияет на ее [значение пластичности](#).

Синапсы импульсной нейронной сети

Место соединения [нейронов](#) импульсной нейронной сети, где происходит передача сигнала, называется *синапсом*. Каждому синапсу может быть присвоено значение веса, на которое изменяется потенциал нейрона при получении спайка от нейрона, связанного с этим синапсом.

В Kaspersky Neuromorphic Platform поддерживаются наборы свойств для следующих синапсов:

- Синапс по модели [AdditiveSTDPDeltaSynapse](#).
- Синапс по модели [DeltaSynapse](#) (далее также "дельта-синапс").
- Синапс по модели [SynapticResourceSTDPDeltaSynapse](#).

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.
- W соответствует функции синаптической пластичности.
- 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 DeltaSynapse

После получения [спайков](#) от пресинаптической [популяции](#) синапсы генерируют сообщение, содержащее значения [синаптических воздействий](#). 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 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: достижением максимально допустимого расстояния ISI_{max} после последнего спайка в ППС, форсированным срабатыванием или приходом 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 ресурса 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_{TSS} соответствует интервалу времени с момента первого спайка в последнем ППС.

$$D_{\max}\left(2 - \frac{|t_{TSS} - ISI_{\max}|}{ISI_{\max}}, -1\right)$$

Формула для расчета изменения стабильности нейрона при дофаминовом вознаграждении

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

Нейроморфный процессор Алтай-2

Нейроморфный процессор Алтай-2 – это сверхбольшая интегральная схема (СБИС) с вычислительной архитектурой, которая имитирует работу биологических сенсорных и нервных систем. Нейроморфный процессор Алтай-2 предназначен для исполнения [импульсных нейронных сетей](#) [или](#) с помощью [компонентов платформы](#) [или](#) с помощью тернарных слоев [пакета ANN2SNN](#). В архитектуре процессора Алтай-2 реализован принцип "вычисления рядом с памятью", позволяющий избежать характерных для архитектуры фон Неймана избыточных энергозатрат на передачу данных между блоком памяти и вычислительным ядром. Нейроморфный процессор Алтай-2 обладает низким энергопотреблением.

Для исполнения импульсной нейронной сети необходимо подключить объединительную плату с нейроморфным процессором Алтай-2 к компьютеру с помощью интерфейсов [PCI-E](#) или [USB](#).

Основной структурной единицей архитектуры нейроморфного процессора Алтай-2 является *ядро*. Ядро объединяет в себе группу нейронов и память для хранения их параметров. Функцию нейронов в ядре выполняет конечный автомат, который моделирует поведение нейронов-интеграторов с утечкой (англ. Leaky Integrate-and-Fire Neuron, далее также "LIF-нейроны").

Синхронизация работы всех нейронов импульсной нейронной сети осуществляется с помощью сигнала [ТИК](#) [?](#), после подачи которого потенциалы всех нейронов последовательно обновляются всеми ядрами и при необходимости испускаются спайки. За тик ядро выполняет 262 144 синаптических операций. При этом частота тика не превышает 2000 Гц. Одна синаптическая операция выполняется за один такт синхросигнала с частотой 600 МГц.

Нейроморфный процессор Алтай-2 имеет регулярную структуру ядер в виде прямоугольной сети, в которой каждое ядро непосредственно связано со своими 4 соседями. Для передачи спайков между нейронами разных ядер реализован механизм маршрутизации таких сигналов по сети ядер. Соответственно, при выполнении функции нейрона к потенциалу моделируемого нейрона могут быть добавлены значения нейронных потенциалов, отправленных нейронами из других ядер.

В нейроморфном процессоре Алтай-2 предусмотрена возможность масштабирования за счет увеличения количества СБИС. По вопросам масштабирования вам нужно обратиться к специалистам АО "Лаборатория Касперского".

[Общие сведения о нейроморфном процессоре Алтай-2](#) [?](#)

В таблице ниже приведены общие сведения о нейроморфном процессоре Алтай-2 и его составляющих:

Общие сведения о процессоре Алтай-2

Свойство		Ограничение
СБИС	Площадь кристалла	6 кв. мм
	Технология изготовления	28 нм
	Максимальное количество нейроморфных ядер	256
	Количество нейроморфных ядер в прототипе	16
	Максимальное количество моделируемых нейронов	131 072
	Количество моделируемых нейронов в прототипе	8192
Ядро	Количество моделируемых нейронов	512
	Количество входных линий	512
	Диапазон значений потенциала нейрона	От -32 768 до 32 767 Потенциал нейрона хранится в виде 16-битной величины.
	Диапазон значений веса синапса	Вес синапса хранится в виде 8-битной беззнаковой величины.
	Вариативность весов на нейрон	4
	Количество синаптических операций	9 Каждая синаптическая связь между моделируемым нейроном и входной линией характеризуется одной из девяти синаптических операций, выбором которой можно управлять воздействием этой синаптической связи на нейрон. Возможны следующие синаптические операции: • Пустая операция. Во время этой операции не происходит никаких изменений. • Прибавить вес N, где N – это номер веса в диапазоне от 1 до 4. • Вычесть вес N, где N – это номер веса в диапазоне от 1 до 4.
	Количество синаптических операций за тик	262 144
	Частота тика	Не более 2000 Гц
	Энергопотребление ядра	Не более 4 мВт
	Объем памяти ядра	88 КБ
	Объем памяти ядра для хранения параметров нейрона	16 КБ

Бэкенды Kaspersky Neuromorphic Platform

Платформа предоставляет несколько независимых друг от друга бэкендов для исполнения [импульсных нейронных сетей](#) на различных вычислительных устройствах.

Бэкенды Kaspersky Neuromorphic Platform запускают импульсные нейронные сети на различных [вычислителях](#), а также выполняют статические проверки корректности построения импульсной нейронной сети и обеспечивают обмен данными между нейронной сетью и [внешней средой](#).

В Kaspersky Neuromorphic Platform поддерживаются следующие типы бэкендов:

- Бэкенды для исполнения импульсных нейронных сетей: однопоточный и многопоточный бэкенды для CPU, бэкенд AltAI SNN.
- Бэкенды для обучения и исполнения искусственных нейронных сетей, преобразованных в импульсные: бэкенды AltAI ANN2SNN.

При необходимости вы можете реализовать и использовать собственные бэкенды.

Для реализации бэкендов не используйте компонент фреймворка, чтобы избежать возникновения циркулярных зависимостей между фреймворком и бэкендами.

Для реализации бэкендов вы можете использовать следующие компоненты платформы:

- *Компонент core*. Предоставляет классы (например, классы проекций и популяций), необходимые для взаимодействия с бэкендом. Компонент поддерживается для языков программирования [C++](#) и [Python](#).
- *Компонент neuron_traits*. Содержит свойства [нейронов](#). Компонент поддерживается для языков программирования [C++](#) и [Python](#).
- *Компонент synapse_traits*. Содержит свойства моделей [синапсов](#). Компонент поддерживается для языков программирования [C++](#) и [Python](#).
- *Компонент meta*. Предоставляет шаблоны общего назначения. Компонент поддерживается для языка программирования [C++](#).

Бэкенды для исполнения импульсных нейронных сетей

Для исполнения импульсных нейронных сетей в Kaspersky Neuromorphic Platform поддерживаются следующие типы бэкендов:

- однопоточный и многопоточный [бэкенды для CPU](#);
- бэкенд для исполнения импульсных нейронных сетей на [нейроморфном процессоре Алтай-2](#) (далее также "бэкенд AltAI SNN").

Однопоточный бэкенд для CPU

Бэкенд работает только в одном потоке. Вы можете использовать однопоточный бэкенд для отладки платформы и работы нейронной сети. Интерфейс к бэкенду реализован в классе [SingleThreadedCPUBackend](#).

На данный момент бэкенд `SingleThreadedCPUBackend` поддерживает [нейроны по моделям](#) AltAILIF, BLIFATNeuron и SynapticResourceSTDPBLIFATNeuron и [синапсы по моделям](#) DeltaSynapse, AdditiveSTDPDeltaSynapse и SynapticResourceSTDPDeltaSynapse.

Многопоточный бэкэнд для CPU

Бэкэнд работает в нескольких потоках одновременно, что обеспечивает повышение скорости расчета нейронной сети. Вы можете использовать многопоточный бэкэнд для обучения и исполнения [импульсной нейронной сети](#) ². Интерфейс к бэкэнду реализован в классе [MultiThreadedCPUBackend](#).

На данный момент бэкэнд `MultiThreadedCPUBackend` поддерживает нейроны по моделям `BLIFATNeuron` и `SynapticResourceSTDPBLIFATNeuron` и синапсы по моделям `DeltaSynapse` и `SynapticResourceSTDPDeltaSynapse`.

Бэкэнд для GPU с поддержкой технологии CUDA

Бэкэнд выполняет вычисления параллельно на множестве ядер графического процессора, что ускоряет процесс обучения и исполнения [импульсной нейронной сети](#) ². Интерфейс к бэкэнду реализован в классе [CUDABackend](#).

Вы можете использовать бэкэнд `CUDABackend` для обучения и исполнения нейронных сетей на устройствах с графическим процессором NVIDIA с поддержкой технологии CUDA.

На данный момент бэкэнд `CUDABackend` поддерживает нейроны по модели `BLIFATNeuron` и синапсы по модели `DeltaSynapse`.

Бэкэнд AltAI SNN

Вы можете использовать бэкэнд AltAI SNN для исполнения импульсных нейронных сетей на [нейроморфном процессоре Алтай-2](#). Интерфейс к бэкэнду реализован в классе `AltAIBackend`. Класс `AltAIBackend` предоставляет стандартные методы, предоставляемые базовым классом [Backend](#).

В качестве типа устройства вы можете выбрать как аппаратный процессор нейроморфного процессора Алтай-2, так и его программный эмулятор. Для аппаратного процессора используется фиксированная конфигурация ядер. При выборе программного эмулятора в качестве типа устройства вы можете [изменить параметры его конфигурации](#).

На рисунке ниже представлена схема взаимодействия компонентов платформы при использовании бэкэнда `AltAIBackend`. Импульсная нейронная сеть создается с помощью [компонентов платформы](#). Обучение нейронной сети необходимо выполнить и обучается методом синаптической пластичности. На данный момент бэкэнд `AltAIBackend` поддерживает нейроны по модели `AltAILIF` и синапсы по модели `DeltaSynapse`, при этом веса синапса должны быть целочисленными и находиться в диапазоне от -255 до 255.

Выполнение обучения импульсной нейронной сети доступно на центральном процессоре методом синаптической пластичности. Если требуется, вы можете реализовать и использовать собственные методы обучения и бэкэнды.

Далее нейронная сеть проверяется на соответствие ограничениям нейроморфного процессора Алтай-2 и переводится в формат, поддерживаемый процессором для ее исполнения. Исполнение импульсной нейронной сети возможно как на программном эмуляторе, так и непосредственно на аппаратном процессоре Алтай-2.

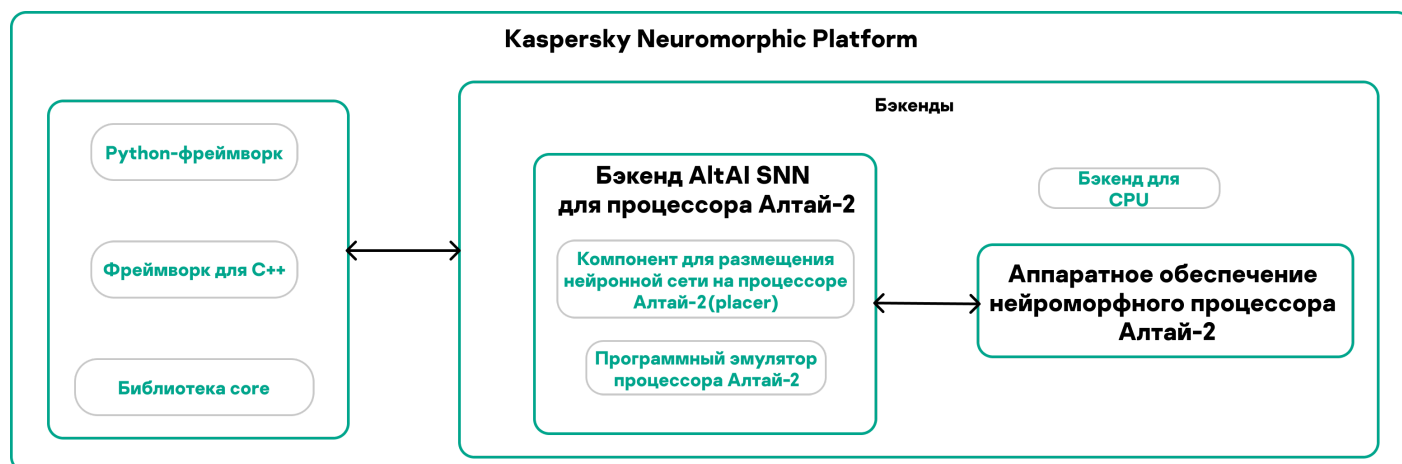


Схема взаимодействия платформы с бэкендом AltAI SNN

Бэкенды для обучения и исполнения искусственных нейронных сетей

В Kaspersky Neuromorphic Platform предусмотрены следующие бэкенды для обучения и исполнения искусственных нейронных сетей [\[2\]](#), преобразованных в импульсные (далее также "бэкенды AltAI ANN2SNN"):

- Бэкенд аппаратного обеспечения нейроморфного процессора Алтай-2;
- Бэкенд программного эмулятора нейроморфного процессора Алтай-2.

Обучение нейронной сети производится обратным распространением ошибки с квантованием весов.

На рисунке ниже представлена схема взаимодействия компонентов платформы при использовании бэкендов AltAI ANN2SNN. Нейронная сеть создается с помощью тернарных слоев пакета [ANN2SNN](#), реализованной как часть Python-фреймворка, и обучается путем обратного распространения ошибки с квантованием весов. Далее нейронная сеть размещается с помощью утилиты `placer` для ее исполнения на нейроморфном процессоре Алтай-2. Во время размещения нейронная сеть проверяется на соответствие ограничениям нейроморфного процессора и переводится в формат, поддерживаемый процессором для ее исполнения.

Бэкенды AltAI ANN2SNN реализованы в пакете, отдельном от бэкендов для CPU и бэкенда AltAI SNN.

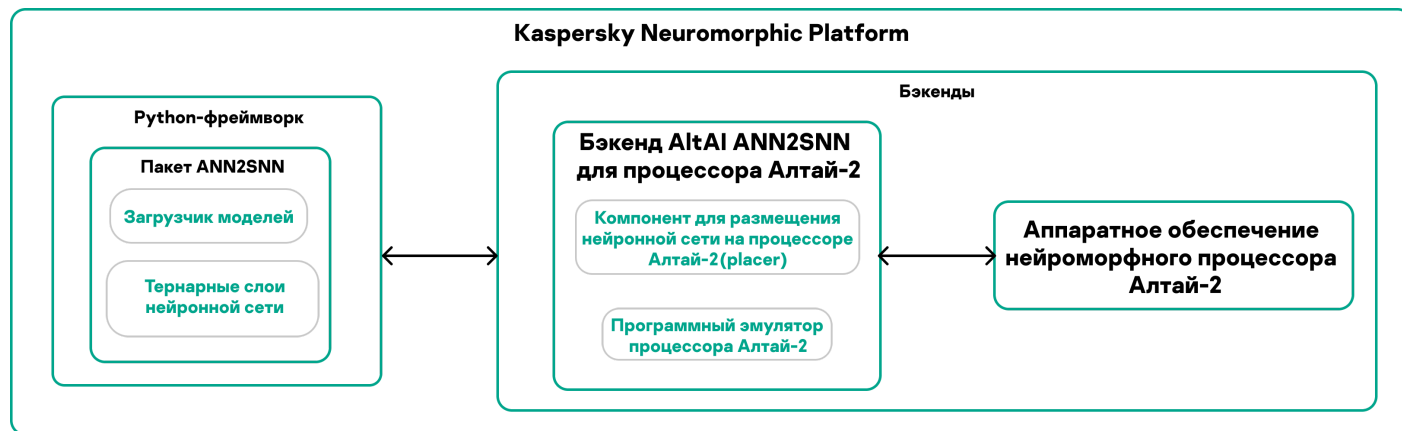


Схема взаимодействия компонентов платформы при использовании пакета AltAI ANN2SNN

Формат описания нейронной сети

В Kaspersky Neuromorphic Platform используется формат [SONATA](#) для описания топологии нейронной сети.

В формате [SONATA](#) нейронная сеть представляется в виде графа, состоящего из вершин, соответствующих нейронам, и ребер, соединяющих вершины графа и соответствующих синапсам. Нейронам и синапсам присваиваются типы соответствующих популяций и проекций. Параметры, назначенные типам нейронов или синапсов, автоматически присваиваются всем нейронам или синапсам этих типов. Кроме того, нейронам, синапсам и их типам могут быть назначены атрибуты, которые определяют различные аспекты нейронной сети (например, положение, тип, параметры модели).

Популяция нейронов состоит из одной или нескольких групп нейронов с единообразной табличной структурой. Популяции сериализуются в HDF5-файлы и связываются между собой CSV-файлом, описывающим типы нейронов и атрибуты, применимые к этим типам. Группы нейронов представляются в виде групп HDF5-файлов, содержащих наборы данных, длины которых соответствуют количеству нейронов в группах. Данные в HDF5-файлах хранятся в бинарном виде.

Аналогично нейронам, синапсы определяются в проекциях, хранящихся в HDF5-файлах. В этих файлах описываются атрибуты для каждого синапса. Проекция синапсов состоит из одной или нескольких групп синапсов с единообразной табличной структурой. Каждый HDF5-файл связан с CSV-файлом, в котором определены типы синапсов и атрибуты, применимые к синапсам этих типов.

На рисунке ниже представлена схема загрузки объекта нейронной сети из формата SONATA и сохранение топологии сети в формат SONATA.



Загрузка и сохранение нейронной сети

Структуры файлов, описывающих группы нейронов и синапсов, см. в [Приложении](#).

Kaspersky Neuromorphic Platform предоставляет функции для загрузки и сохранения нейронной сети в формате SONATA. Функции реализованы в пространстве имен [sonata](#), предоставляемом как во [фреймворке для C++](#), так и в [Python-фреймворке](#).

Форматы данных, обрабатываемые платформой

Для хранения [входных](#) и [выходных данных](#) в виде временного ряда используется базовый формат Kaspersky Neuromorphic Platform (далее также "формат KNP"). Файл базового формата содержит следующую информацию:

- Заголовок файла, содержащий следующую информацию:
 - версию формата файла;
 - массив идентификаторов UID отправителей данных для каждого шага исполнения нейронной сети (опционально);
 - массив тегов.
- Входные или выходные данные представляют собой временной ряд, состоящий из списка структур со следующей информацией:
 - шагом исполнения нейронной сети, на котором были сгенерированы спайки;
 - индексами нейронов, сгенерировавших спайки на этом шаге.

Структуру и пример файла базового формата см. в [Приложении](#).

С помощью функций Kaspersky Neuromorphic Platform вы можете преобразовать файлы базового формата в сообщения [SpikeMessage](#).

Сообщения формата KNP сериализуются в следующие нотации:

- [HDF5](#) – основная нотация для хранения данных в бинарном виде;
- [JSON](#) – нотация объектов JavaScript, которая может быть использована для отладки работы нейронной сети.

Фреймворк для C++ Kaspersky Neuromorphic Platform предоставляет пространство имен [native](#), в котором реализованы функции для загрузки сообщений [SpikeMessage](#) из файлов базового формата, а также их сохранения в файлы базового формата.

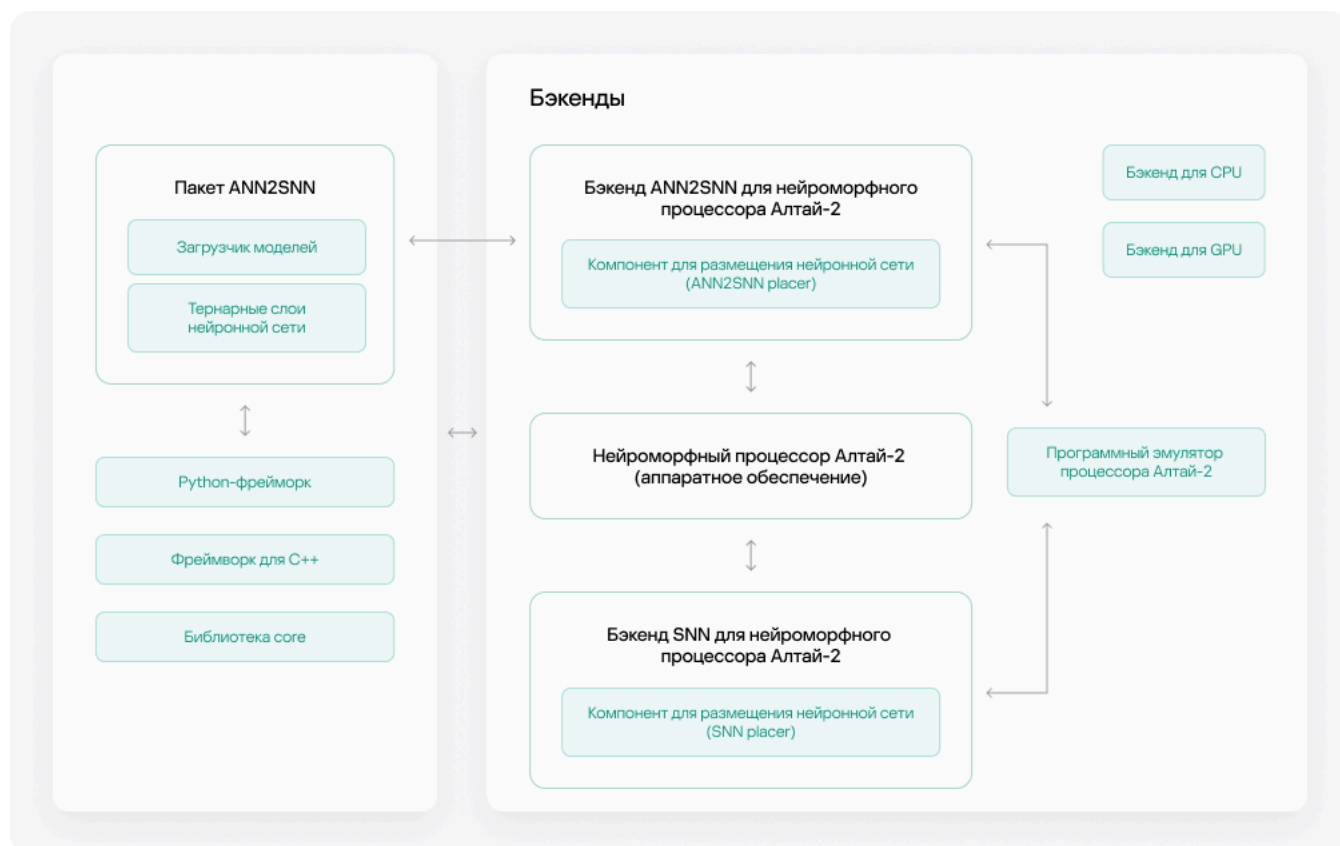
Помимо базового формата входные данные также могут храниться и быть переданы в виде CSV-файлов, в которых содержатся индексы нейронов, сгенерировавших спайки. CSV-файлы с входными данными могут быть переданы в преобразователь данных [IndexConverter](#) для их дальнейшей обработки в каналах ввода.

Архитектура Kaspersky Neuromorphic Platform

Kaspersky Neuromorphic Platform включает в себя следующие компоненты:

- [Python-фреймворк](#). Предназначен для создания [импульсных нейронных сетей](#) ^[2], а также преобразования [искусственных нейронных сетей](#) ^[2] в импульсные. Созданные импульсные нейронные сети обучаются методами синаптической пластичности и могут быть исполнены на центральном процессоре, нейроморфном процессоре Алтай-2 или его программном эмуляторе. Искусственные нейронные сети, преобразованные в импульсные, обучаются методом обратного распространения ошибки с учетом [ограничений нейроморфного процессора Алтай-2](#). Полученные в результате нейронные сети адаптированы для исполнения на нейроморфном процессоре Алтай-2 или его программном эмуляторе.
Python-фреймворк включает в себя модули, функциональность которых пересекается с функциональностью фреймворка для C++, а также пакет [ANN2SNN](#), необходимый для преобразования искусственных нейронных сетей в импульсные и их исполнения на [бэкенде AltAI ANN2SNN](#) ^[2].
- [Фреймворк для C++](#). Предназначен для создания импульсных нейронных сетей и их обучения методами синаптической пластичности. Полученные в результате нейронные сети могут исполняться на центральном процессоре, нейроморфном процессоре Алтай-2 или его программном эмуляторе.
- [Библиотека core](#). Предоставляет классы для взаимодействия с бэкендом и выполнения прикладных задач.
- [Бэкенды](#). Запускают обучение и исполнение импульсной нейронной сети на различных вычислительных устройствах (например, на нейроморфном процессоре Алтай-2). Платформа предоставляет несколько независимых друг от друга бэкендов, каждый из которых позволяет обучать и исполнять импульсные нейронные сети на соответствующем типе устройства.

На рисунке ниже представлена схема взаимодействия компонентов Kaspersky Neuromorphic Platform.



Архитектура Kaspersky Neuromorphic Platform

Изменение параметров конфигурации программного эмулятора

Если при использования бэкенда [AltAI SNN](#) вы выбрали программный эмулятор в качестве типа устройства, вы можете изменить параметры конфигурации ядер [нейроморфного процессора Алтай-2](#).

Чтобы изменить параметры конфигурации программного эмулятора:

1. Измените параметры конфигурации ядер нейроморфного процессора Алтай-2 с помощью метода программного эмулятора `load_core_grid_params()`.

Метод `load_core_grid_params()` необходимо вызвать перед инициализацией бэкенда.

2. Измените расположение портов ввода и вывода с помощью функции программного эмулятора `make_standart_north_port()`.

Общие принципы работы Kaspersky Neuromorphic Platform

Этот раздел содержит описание общих принципов работы Kaspersky Neuromorphic Platform.

Обмен данными в Kaspersky Neuromorphic Platform

При [работе с исходным кодом](#) необходимо учитывать особенности механизма обмена данными в платформе.

Kaspersky Neuromorphic Platform обеспечивает обмен данными между популяциями и проекциями, а также между другими объектами платформы и внешними объектами.

Обмен данными между бэкендом и внешней средой

Передача данных между бэкендом и внешней средой осуществляется посредством сообщений, содержащих спайки. На бэкенде сообщения со спайками принимают входные проекции. Во внешнюю среду сообщения со спайками отправляются по подпискам.

Сообщения со спайками могут быть переданы в синхронном режиме (по запросу пользователя) или по сигналу бэкенда. Для получения спайков в синхронном режиме вызывается блокирующий метод источника данных.

Если требуется, вы можете реализовать и использовать асинхронный режим для обмена данными между бэкендом и внешней средой.

Обмен данными между нейронной сетью и внешней средой

Из внешней среды в нейронную сеть данные поступают через каналы ввода. Каналы ввода используются для преобразования полученных данных в сообщения, содержащие спайки.

Типовой цикл получения нейронной сетью данных от внешней среды включает следующие шаги:

1. В соответствии с заданной функцией преобразователь данных [IndexConverter](#) или [SequenceConverter](#) преобразует данные, полученные от внешней среды, в сообщения [SpikeMessage](#).
2. Канал ввода [InputChannel](#) получает преобразованные сообщения от преобразователя данных.
3. Точка подключения [MessageEndpoint](#), связанная с каналом ввода `InputChannel`, отправляет спайки во входную проекцию нейронной сети [Network](#).

Во внешнюю среду нейронная сеть передает спайки, преобразованные в данные, через каналы вывода, подключенные к указанным популяциям. Типовой цикл передачи данных нейронной сетью во внешнюю среду включает следующие шаги:

1. Точка подключения `MessageEndpoint` подписывается на [сообщения](#) отправителя.
2. Точка подключения `MessageEndpoint` отправляет сообщения от связанной популяции.
3. Шина сообщений `MessageBus` доставляет сообщения от отправителя к точке подключения `MessageEndpoint`, связанной с каналом вывода `OutputChannel1`.
4. Точка подключения `MessageEndpoint`, связанная с каналом вывода `OutputChannel1`, получает сообщения.
5. Канал вывода `OutputChannel1` получает сообщения и преобразовывает их в нужный формат данных.

Если вы загрузили [модель](#), хранящую нейронную сеть, каналы ввода и вывода, в исполнитель моделей `ModelExecutor`, то при ее исполнении на бэкенде автоматически осуществляется передача отправленных сообщений до точки подключения через шину сообщений.

Передача данных (например, потенциала мембраны) возможна также через [подсистему мониторинга](#).

Если требуется, вы можете реализовать и использовать собственные объекты для обмена данными между нейронной сетью и внешней средой.

Обмен данными между популяциями и проекциями

Обмен данными между популяциями и проекциями осуществляется через [сообщения](#). Для обмена сообщениями популяции и проекции требуется предварительно загрузить на [бэкенд](#).

На обмене сообщений между популяциями и проекциями основывается [работа бэкенда для CPU](#).

Работа бэкенда для CPU

Работа бэкенда для CPU основывается на [обмене сообщениями](#) между проекциями и популяциями. Объекты платформы также могут обмениваться сообщениями с объектами [внешней среды](#)². Для обмена сообщений бэкенд предоставляет шину сообщений `MessageBus`.

Типовой цикл работы бэкенда включает следующие шаги:

1. При инициализации бэкенда загруженные проекции синапсов подписываются на сообщения от связанных популяций, а загруженные популяции подписываются на сообщения от связанных проекций. При необходимости вы можете вручную подписать популяции и проекции друг на друга с помощью метода `Backend::subscribe()` или `MessageEndpoint::subscribe()`.
2. Каналы ввода `InputChannel1` отправляют данные в шину сообщений `MessageBus`.
3. Бэкенд вычисляет функцию [нейронов](#)² популяций.
4. Нейроны популяций генерируют [спайки](#)². Условия генерации спайков определяются [типами нейронов](#) популяции и значениями их атрибутов.

5. Популяции нейронов отправляют сообщения (например, сообщения со спайками) в шину сообщений `MessageBus` через точки подключения `MessageEndpoint`.
На сообщения популяции может быть подписано более одной проекции.
6. Шина сообщений `MessageBus` в соответствии с подписками отправляет данные, полученные от популяций, в [проекции](#) или каналы вывода [OutputChannel1](#). На сообщения, содержащие спайки, может быть подписано более одной проекции.
7. Бэкенд вычисляет [синаптические воздействия](#) ^[2].
8. Проекция синапсов отправляет синаптические воздействия с результатами вычислений в шину сообщений `MessageBus` с помощью точек подключения `MessageEndpoint`.
9. Шина сообщений `MessageBus` отправляет данные, полученные от проекций, в связанные популяции нейронов.

Обработка сообщений по подписке

[Подписка](#) обеспечивает фильтрацию [сообщений](#). Фильтрация сообщений осуществляется в соответствии с уникальными идентификаторами отправителей и получателей, определенных в объекте подписки. Сообщения хранятся в подписке до их запроса получателем или до их удаления из подписки.

Типовой цикл обработки сообщений по подписке включает следующие шаги:

1. Отправка сообщений через точки подключения.

Объекты платформы отправляют сообщения с помощью метода [MessageEndpoint::send_message\(\)](#).

2. Маршрутизация и фильтрация сообщений.

После вызова метода [route_messages\(\)](#) шина сообщений `MessageBus` распределяет сообщения по подпискам в соответствии с определенными в них уникальными идентификаторами отправителей и получателей сообщений.

3. Доставка сообщений до получателей.

Получатель запрашивает сообщения с помощью методов [receive_all_messages\(\)](#) и [unload_messages\(\)](#) класса `MessageEndpoint`.

Установка и удаление платформы

Этот раздел содержит пошаговые инструкции по установке и удалению Kaspersky Neuromorphic Platform.

Для разработки прикладных решений на Linux вы можете установить платформу одним из следующих способов:

- [Установить deb-пакеты](#).
- [Установить пакеты](#) для разработки прикладных решений на языке программирования Python.

При работе с исходным кодом платформы на Linux или в ОС Windows вы можете [собрать платформу или прикладное решение на C++](#). Сборка платформы может быть использована для разработки Kaspersky Neuromorphic Platform.

Установка deb-пакетов

На компьютере должно быть установлено программное обеспечение, которое указано в [аппаратных и программных требованиях](#).

Установка deb-пакетов выполняется на Linux.

Для разработки прикладных решений необходимо установить следующие пакеты:

- Пакеты из архива `pkg-deb.zip`:
 - `knp-cpu-single-threaded-backend_<knp_version>_amd64.deb` – deb-пакет с динамической библиотекой однопоточного бэкенда для CPU;
 - `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb` – deb-пакет с динамической библиотекой многопоточного бэкенда для CPU;
 - `knp-gpu-cuda-backend_<knp_version>_amd64.deb` – deb-пакет с библиотекой бэкенда для GPU с поддержкой технологии CUDA;
 - `knp-cpp-framework_<knp_version>_amd64.deb` – deb-пакет с динамической библиотекой фреймворка для C++;
 - `knp-cpu-single-threaded-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb`;
 - `knp-cpu-multi-threaded-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb`;
 - `knp-gpu-cuda-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-gpu-cuda-backend_<knp_version>_amd64.deb`;
 - `knp-cpp-framework-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpp-framework_<knp_version>_amd64.deb`;
 - `knp-python3-framework_<knp_version>_amd64.deb` – deb-пакет с Python-фреймворком для работы на бэкенде AltAI SNN;
 - `knp-examples_<knp_version>_amd64.deb` – deb-пакет с примерами использования Kaspersky Neuromorphic Platform;
 - `knp-documentation_<knp_version>_all.deb` – deb-пакет со справочником API.
- `knp-altai-backend_<motiv_version>_amd64.deb` – deb-пакет с динамической библиотекой [бэкенда AltAI SNN](#);
- `knp-altai-backend-dev_<motiv_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-altai-backend_<motiv_version>_amd64.deb`;
- `knp_ann2snn.deb` – deb-пакет с пакетом ANN2SNN, динамической библиотекой [бэкенда AltAI ANN2SNN](#) для [нейроморфного процессора Алтай-2](#) и компонентом для размещения нейронной сети на нейроморфном процессоре Алтай-2.

где:

- `<knp_version>` – версия Kaspersky Neuromorphic Platform;
- `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-2 и пакета ANN2SNN.

Чтобы установить *deb*-пакеты:

1. Скачайте архив `pkg-deb.zip` и пакеты с [репозитория платформы](#).
2. Распакуйте архив `pkg-deb.zip`.
3. Для установки *deb*-пакетов с динамическими библиотеками бэкендов для CPU, бэкенда для GPU и бэкенда `AltAI SNN` запустите команды:

```
sudo dpkg -i <path-to-deb-package>/knp-cpu-single-threaded-backend_<knp_version>_amd64.deb
sudo dpkg -i <path-to-deb-package>/knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb
sudo dpkg -i <path-to-deb-package>/knp-gpu-cuda-backend_<knp_version>_amd64.deb
sudo dpkg -i <path-to-deb-package>/knp-altai-backend_<motiv_version>_amd64.deb
```

где `<path-to-deb-package>` – путь к *deb*-пакету.

Во время установки пакета `knp-altai-backend_<motiv_version>_amd64.deb` примите Условия использования бэкендов для нейроморфного процессора Алтай-2 и пакета `ANN2SNN`. Согласие с Условиями использования является обязательным условием для установки пакета. Если вы не принимаете Условия использования, установка пакета будет прервана.

4. Для установки *deb*-пакета с динамической библиотекой фреймворка для C++ запустите команду:

```
sudo dpkg -i <path-to-deb-package>/knp-cpp-framework_<knp_version>_amd64.deb
```

5. Для установки *dev-deb* пакетов запустите команды:

```
sudo dpkg -i <path-to-deb-package>/knp-cpu-multi-threaded-backend-dev_<knp_version>_amd64.deb
sudo dpkg -i <path-to-deb-package>/knp-cpu-single-threaded-backend-dev_<knp_version>_amd64.deb
sudo dpkg -i <path-to-deb-package>/knp-gpu-cuda-backend-dev_<knp_version>_amd64.deb
sudo dpkg -i <path-to-deb-package>/knp-altai-backend-dev_<motiv_version>_amd64.deb
sudo dpkg -i <path-to-deb-package>/knp-cpp-framework-dev_<knp_version>_amd64.deb
```

Во время установки пакета `knp-altai-backend-dev_<motiv_version>_amd64.deb` примите Условия использования бэкендов для нейроморфного процессора Алтай-2 и пакета `ANN2SNN`. Согласие с Условиями использования является обязательным условием для установки пакета. Если вы не принимаете Условия использования, установка пакета будет прервана.

6. Для установки *deb*-пакета с Python-фреймворком для работы на бэкенде `AltAI SNN` запустите команду:

```
sudo dpkg -i <path-to-deb-package>/knp-python3-framework_<knp_version>_amd64.deb
```

7. Для установки *deb*-пакета с пакетом `ANN2SNN` и динамической библиотекой бэкенда `AltAI ANN2SNN` запустите команду:

```
sudo dpkg -i <path-to-deb-package>/knp_ann2snn.deb
```

Во время установки пакет `knp_ann2snn.deb` примите Условия использования бэкендов для нейроморфного процессора Алтай-2 и пакета `ANN2SNN`. Согласие с Условиями использования является обязательным условием для установки пакета. Если вы не принимаете Условия использования, установка пакета будет прервана.

8. Для установки deb-пакета с примерами использования платформы запустите команду:

```
sudo dpkg -i <path-to-deb-package>/knp-examples_<knp_version>_amd64.deb
```

9. Для установки deb-пакета со справочником API запустите команду:

```
sudo dpkg -i <path-to-deb-package>/knp-documentation_<knp_version>_all.deb
```

Установка пакетов для разработки на языке программирования Python

На компьютере должно быть установлено программное обеспечение, которое указано в [аппаратных и программных требованиях](#).

Установка пакетов выполняется на Linux.

Для разработки на языке программирования Python необходимо установить следующие deb- или whl-пакеты:

- deb-пакеты:
 - `knp-python3-framework_<knp_version>_amd64.deb` – deb-пакет с Python-фреймворком для работы на [бэкенде AltAI SNN](#) из архива `pkg-deb.zip`.
 - `knp_ann2snn.deb` – deb-пакет с пакетом `ANN2SNN`, динамической библиотекой [бэкенда AltAI ANN2SNN](#) для [нейроморфного процессора Алтай-2](#) и компонентом для размещения нейронной сети на нейроморфном процессоре Алтай-2.
- whl-пакеты:
 - `knp-<knp_version>-py3-none-any.whl` – whl-пакет с Python-фреймворком для работы на бэкенде `AltAI SNN` из архива `pkg-python.zip`.
 - `knp_ann2snn_x86_64-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом `ANN2SNN`, динамической библиотекой бэкенда `AltAI ANN2SNN` для [нейроморфного процессора Алтай-2](#) и компонентом для размещения нейронной сети на нейроморфном процессоре Алтай-2.

Этот пакет устанавливается на компьютеры с процессором архитектуры x86.

- `knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом `ANN2SNN` и динамической библиотекой бэкенда `AltAI ANN2SNN` для нейроморфного процессора Алтай-2.

Этот пакет устанавливается на компьютеры с процессором архитектуры ARM.

где:

- `<knp_version>` – версия Kaspersky Neuromorphic Platform;
- `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-2 и пакета `ANN2SNN`.

Чтобы установить deb-пакеты для разработки на языке программирования Python:

1. Скачайте архив `pkg-deb.zip` и пакет `knp_ann2snn.deb` с [репозитория платформы](#).
2. Распакуйте архив.

3. Для установки deb-пакета с Python-фреймворком для работы на бэкенде AltAI SNN запустите команду:

```
sudo dpkg -i <path-to-deb-package>/knp-python3-framework_<knp_version>_amd64.deb
```

где:

- `<path-to-deb-package>` – путь к deb-пакету;
- `<knp_version>` – версия Kaspersky Neuromorphic Platform.

4. Для установки deb-пакета с пакетом ANN2SNN и динамической библиотекой бэкенда AltAI ANN2SNN запустите команду:

```
sudo dpkg -i <path-to-deb-package>/knp_ann2snn.deb
```

Во время установки пакета `knp_ann2snn.deb` примите Условия использования бэкендов для нейроморфного процессора Алтай-2 и пакета ANN2SNN. Согласие с Условиями использования является обязательным условием для установки пакета. Если вы не принимаете Условия использования, установка пакета будет прервана.

Для установки whl-пакетов необходимо использовать менеджер пакетов `pip3`.

Чтобы установить whl-пакеты:

1. Скачайте архив `pkg-python.zip` и whl-пакеты с [репозитория платформы](#).
2. Распакуйте архив.
3. Для установки whl-пакета с Python-фреймворком для работы на бэкенде AltAI SNN запустите команду:

```
pip3 install --break-system-packages knp-<knp_version>-py3-none-any.whl
```

где `<knp_version>` – версия Kaspersky Neuromorphic Platform.

4. Для установки whl-пакета с пакетом ANN2SNN и динамической библиотекой бэкенда AltAI ANN2SNN выполните одно из следующих действий:

- Если требуется установить пакет на компьютере с процессором архитектуры x86, запустите команду:

```
pip3 install --break-system-packages knp_ann2snn_x86_64-<motiv_version>-py3-none-any.whl
```

- Если требуется установить пакет на компьютере с процессором архитектуры ARM, запустите команду:

```
pip3 install --break-system-packages knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl
```

где `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-2 и пакета ANN2SNN.

Удаление платформы

В зависимости от способа установки вы можете удалить платформу следующими способами:

- Удалить deb-пакеты.
- Удалить пакеты для разработки прикладных решений на языке программирования Python.
- Удалить директорию, в которую был распакован архив с исходным кодом платформы.

Удаление deb-пакетов

Для удаления deb-пакетов рекомендуется использовать пакетный менеджер `apt`.

Чтобы удалить deb-пакеты:

1. Для удаления deb-пакетов с динамической библиотекой бэкендов для CPU, бэкендом для GPU и бэкендом `AltAI SNN` запустите команды:

```
sudo apt purge knp-cpu-single-threaded-backend
sudo apt purge knp-cpu-multi-threaded-backend
sudo apt purge knp-gpu-cuda-backend
sudo apt purge knp-altai-backend
```

2. Для удаления deb-пакета с динамической библиотекой фреймворка для C++ запустите команду:

```
sudo apt purge knp-cpp-framework
```

3. Для удаления dev-deb пакетов запустите с root-правами команды:

```
sudo apt purge knp-cpu-multi-threaded-backend-dev
sudo apt purge knp-cpu-single-threaded-backend-dev
sudo apt purge knp-gpu-cuda-backend-dev
sudo apt purge knp-altai-backend-dev
sudo apt purge knp-cpp-framework-dev
```

4. Для удаления deb-пакета с Python-фреймворком для работы на бэкенде `AltAI SNN` запустите команду:

```
sudo apt purge knp-python3-framework
```

5. Для удаления deb-пакета с пакетом `ANN2SNN` и динамической библиотекой бэкенда `AltAI ANN2SNN` запустите команду:

```
sudo apt purge knp_ann2snn
```

6. Для удаления deb-пакета с примерами использования платформы запустите команду:

```
sudo apt purge knp-examples
```

7. Для удаления deb-пакета со справочником API запустите команду:

```
sudo apt purge knp-documentation
```

Удаление пакетов для разработки прикладных решений на Python

Для удаления deb-пакетов, используемых для разработки прикладных решений на Python, рекомендуется использовать пакетный менеджер `apt`.

Чтобы удалить deb-пакеты:

1. Для удаления deb-пакета с Python-фреймворком для работы на бэкенде `AltAI SNN` запустите команду:

```
sudo apt purge knp-python3-framework
```

2. Для удаления deb-пакета с пакетом `ANN2SNN` и динамической библиотекой бэкенда `AltAI ANN2SNN` запустите команду:

```
sudo apt purge knp_ann2snn
```

Для удаления whl-пакетов необходимо использовать пакетный менеджер `pip3`.

Чтобы удалить whl-пакеты:

1. Для удаления whl-пакета с Python-фреймворком для работы на бэкенде `AltAI SNN` запустите команду:

```
pip3 uninstall --break-system-packages knp-<knp_version>-py3-none-any.whl
```

где `<knp_version>` – версия Kaspersky Neuromorphic Platform.

2. Для удаления whl-пакета с пакетом `ANN2SNN` и динамической библиотекой бэкенда `AltAI ANN2SNN` выполните одно из следующих действий:

- Если требуется удалить пакет с компьютера с процессором архитектуры x86, запустите команду:

```
pip3 uninstall --break-system-packages knp_ann2snn_x86_64-<motiv_version>-py3-none-any.whl
```

- Если требуется удалить пакет с компьютера с процессором архитектуры ARM, запустите команду:

```
pip3 uninstall --break-system-packages knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl
```

где `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-2 и пакета `ANN2SNN`.

Удаление исходного кода платформы

Чтобы удалить исходный код платформы,

удалите локальный репозиторий платформы или рабочую директорию, в которую был [распакован архив](#) с исходным кодом платформы.

Сценарий сборки проекта платформы или прикладного решения на C++

Вы можете собрать платформу или прикладное решение на C++ из исходного кода. Для сборки прикладного решения вы можете добавить платформу как зависимость с помощью функции `add_subdirectory` в CMake или с помощью менеджера зависимостей (например, CPM.cmake).

При разработке прикладных решений на Linux вы можете [установить нужные deb-пакеты](#) вместо сборки прикладного решения.

На компьютере должно быть установлено программное обеспечение, которое указано в [аппаратных и программных требованиях](#).

Сборку под Linux вы можете выполнить в Docker-контейнере. Для этого вы можете использовать Docker-образ `knp-build-image`, входящий в [комплект поставки платформы](#).

Сценарий сборки платформы или прикладного решения состоит из следующих этапов:

1 Подготовка к сборке

Получите исходный код платформы. Вы можете получить исходный код платформы одним из следующих способов:

- [Склонировать репозиторий платформы](#).
- [Скачать архив с исходным кодом](#) из репозитория платформы и распаковать его в рабочую директорию.

После получения исходного кода платформы установите библиотеку [Boost](#) нужной версии.

При работе в ОС Windows рекомендуется установить библиотеку, [скомпилированную](#) предварительно для Visual Studio. Например, для Visual Studio 2022 версия компилятора – 14.3. Соответственно для установки библиотеки Boost, скомпилированной для Visual Studio, необходимо скачать исполняемый файл с именем в формате `boost_<версия Boost>-msvc-14.3-<разрядность ОС Windows>.exe`.

2 Конфигурация сборки платформы

В ОС Windows для CMake необходимо указать директорию, в которой установлена библиотека Boost. Вы можете это сделать одним из следующих способов:

- Указать путь к установленной библиотеке Boost в параметре `Boost_ROOT` в файле `CMakePresets.json`, расположенном в корневой директории с исходным кодом платформы.

CMake кеширует значения переменных. Если после определения параметра `Boost_ROOT` CMake не может найти библиотеку Boost, необходимо заново [сгенерировать кеш CMake](#).

- Создать переменную окружения `Boost_ROOT`.

- Определить путь к установленной библиотеке Boost как параметр вызова CMake. Для этого при конфигурации сборки используйте следующий параметр:

```
-DBOOST_ROOT=<путь к установленной библиотеке Boost>
```

В случае сборки платформы с Python-фреймворком изучите [особенности сборки](#).

Если вы собираете платформу на Linux, выполните конфигурацию сборки с помощью системы сборки CMake. Например, вы можете использовать следующую команду:

```
cmake -S. -Bbuild --preset <пресет для сборки> -DCMAKE_C_COMPILER=<компилятор> -DCMAKE_CXX_COMPILER=<компилятор>
```

где:

- <пресет для сборки> – имя файла пресета, описывающего параметры сборки.

Вы можете указать один из пресетов, описанных в файле `CMakePresets.json`, расположенном в корневой директории с исходным кодом платформы. Описание пресетов, предоставляемых Kaspersky Neuromorphic Platform, см. в [Приложении](#).

Опция `--preset <пресет для сборки>` при выполнении конфигурации сборки не является обязательной.

- <компилятор> – компиляторы для языка программирования C и C++.

При конфигурации сборки вы можете задать [параметры конфигурации платформы](#).

Подробнее о конфигурации сборки с помощью CMake см. [документацию CMake](#).

Если вы собираете платформу в ОС Windows, [выполните конфигурацию сборки](#) в Visual Studio. Для этого откройте проект, выбрав нужный файл `CMakeLists.txt`, и выполните настройку кеша. Для сборки платформы необходимо выбрать файл `CMakeLists.txt`, расположенный в корневой директории с исходным кодом платформы.

Первая конфигурация сборки с помощью CMake может выполняться долго и завершаться с сетевыми ошибками. В случае возникновения сетевых ошибок выполните конфигурацию сборки еще раз.

3 Запуск сборки платформы

Запустите сборку платформы или прикладного решения. Подробнее о запуске сборки проектов см. [документацию CMake](#) или документацию используемой интегрированной среды разработки.

Для запуска сборки на Linux вы можете использовать команду:

```
cmake -B build .
```

Для запуска сборки в ОС Windows в Visual Studio на вкладке **Сборка** выберите **Собрать все**.

4 Сборка прикладного решения

Для сборки прикладного решения создайте файл `CMakeLists.txt` и [опишите в нем процесс сборки](#). Добавьте платформу как зависимость, указав директорию с платформой с помощью функции `add_subdirectory`. Укажите исполняемый файл с помощью функции `add_executable`. Добавьте библиотеки платформы, которые необходимо подключить к проекту, с помощью функции `target_link_libraries`. Вы можете использовать следующие библиотеки:

- [neuron traits](#)
- [synapse traits](#)
- [core](#)
- [meta](#)
- [backends](#)
- [framework](#)

Выполните конфигурацию сборки для прикладного решения. В случае сборки в [OC Windows](#) при открытии проекта выберите файл [CMakeLists.txt](#) прикладного решения. Запустите сборку прикладного решения.

Загрузка и распаковка архива с исходным кодом платформы

Чтобы загрузить и распаковать архив с исходным кодом платформы:

1. Скачайте [архив с исходным кодом](#) из репозитория платформы.
2. Распакуйте в рабочую директорию архив платформы.

Например, вы можете распаковать архив платформы с помощью следующей команды:

```
tar xf <путь_к_архиву_платформы> --directory <путь_к_рабочей_директории>
```

Определение параметра Boost_ROOT в OC Windows

Перед сборкой платформы или прикладного решения в OC Windows требуется определить параметр `Boost_ROOT`. В качестве значения параметра необходимо указать путь к [установленной библиотеке Boost](#). Для этого вам нужно указать его в параметре `Boost_ROOT` в файле `CMakePresets.json`, расположенном в корневой директории с исходным кодом платформы.

Вы также можете создать переменную окружения для параметра `Boost_ROOT`.

Чтобы создать переменную окружения в OC Windows:

1. В OC Windows нажмите комбинацию клавиш **Windows+R**.
2. Введите `sysdm.cpl` и нажмите на клавишу **Enter**.
3. В открывшемся окне выберите вкладку **Дополнительно** и нажмите на кнопку **Переменные среды**.
4. В открывшемся окне создайте переменную `Boost_ROOT` и укажите путь к библиотеке Boost в качестве значения переменной.

Особенности сборки платформы с Python-фреймворком

Сборка платформы с Python-фреймворком имеет следующие особенности:

- Работа с Python-фреймворком на устройстве возможна, если на устройстве установлены Python 3.12 и выше и библиотека Boost, версия которой соответствует версии Python.
- При необходимости для работы с Python-фреймворком вы можете использовать виртуальное окружение. Для создания виртуального окружения перейдите в рабочую директорию и выполните команду:

```
python3 -m venv venv
```

Для подключения к созданному виртуальному окружению выполните команду:

```
source venv/bin/activate
```

- Платформа автоматически собирается с Python-фреймворком, если на устройстве есть инструмент stubgen. Если инструмент stubgen на устройстве отсутствует, вы можете выполнить одно из следующих действий:

- Установите пакет mypy для доступа к инструменту stubgen. Для этого выполните команду:

```
pip install mypy
```

- Включите сборку платформы с Python-фреймворком с помощью параметра конфигурации сборки [KNP_PYTHON_FRAMEWORK_BUILD](#).
- Если на устройстве есть инструмент stubgen, при запуске сборки автоматически собирается whl-пакет с Python-фреймворком. Если stubgen на устройстве отсутствует, вы можете включить сборку whl-пакета с помощью параметра конфигурации сборки [KNP_PYTHON_BUILD_WHEEL](#).

Конфигурация сборки платформы или прикладного решения в ОС Windows

В ОС Windows вы можете выполнить конфигурацию сборки платформы или прикладного решения с помощью Visual Studio.

Чтобы выполнить конфигурацию сборки платформы или прикладного решения в ОС Windows:

1. В ОС Windows запустите Visual Studio.
2. На вкладке **Файл** выберите **Открыть** → **CMake**.

3. В открывшемся окне выберите корневой файл `CMakeLists.txt`.

В случае сборки прикладного решения выберите файл `CMakeLists.txt` прикладного решения. При сборке платформы выберите файл `CMakeLists.txt`, расположенный в корневой директории с исходным кодом платформы.

4. На вкладке **Проект** выберите **Настроить кэш** или **Настроить kpr**.

CMake начнет работы по конфигурации сборки.

В случае возникновения сетевых ошибок выполните конфигурацию сборки еще раз.

После того, как CMake завершит работы по конфигурации, вам нужно запустить сборку.

Настройка параметров сборки проекта прикладного решения на C++

Для сборки прикладного решения необходимо указать параметры сборки в файле `CMakeLists.txt`. Файл `CMakeLists.txt` должен содержать следующие команды:

`cmake_minimum_required(VERSION 3.25)`

Указание минимальной поддерживаемой версии CMake. Для сборки проекта на базе Kaspersky Neuromorphic Platform требуется CMake версии не ниже 3.25.

`project(<название_проекта>)`

Указание названия проекта.

`set(CMAKE_CXX_STANDARD 17)`

Указание C++ 17 в качестве используемого языкового стандарта.

`add_subdirectory(<директория_с_исходным_кодом_платформы>)`

Указание директории, включающей программу, сборку которой требуется выполнить.

`add_executable("${PROJECT_NAME}" <файл_программы>)`

Указание исполняемого файла.

`target_link_libraries("${PROJECT_NAME}" PRIVATE <подключаемые_библиотеки_платформы>)`

Определение библиотек Kaspersky Neuromorphic Platform, которые будут подключены к вашему проекту. Вы можете подключить библиотеки `backends`, `neuron trits`, `synapse traits`, `core`, `meta` и `framework`.

Перед сборкой проекта в ОС Windows необходимо [определить параметр](#) `Boost_ROOT`.

Подробнее о конфигурации сборки с помощью CMake см. [документацию CMake](#).

Чтобы настроить параметры сборки проекта прикладного решения на C++:

1. Перейдите в рабочую директорию.

2. В рабочей директории создайте файл `CMakeLists.txt`.

3. Откройте созданный файл в любом текстовом редакторе и укажите параметры сборки проекта.

Пример CMakeLists.txt

```
cmake_minimum_required(VERSION 3.25)
# 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("${PROJECT_NAME}" main.cpp)
# Подключает фреймворк и библиотеку однопоточного бэкенда для CPU к проекту digits-recognition.
target_link_libraries("${PROJECT_NAME}" PRIVATE KNP::BaseFramework::Core KNP::Backends::CPUSingleThreaded)
```

При необходимости вы можете загрузить фреймворк динамически.

4. Сохраните изменения в файле CMakeLists.txt.

Компоненты платформы для использования на C++

Этот раздел содержит описание компонентов для использования Kaspersky Neuromorphic Platform на языке программирования C++.

Библиотека neuron_traits

Библиотека `neuron_traits` представляет собой набор атрибутов [нейронов](#) ². Вы можете использовать атрибуты нейронов, представленных в библиотеке, например, при построении популяций.

На текущий момент библиотека поддерживает наборы свойств для следующих нейронов:

- Нейрон по модели [AltAILIF](#).
- Нейрон по модели [BLIFATNeuron](#).
- Нейрон по модели [SynapticResourceSTDPBLIFATNeuron](#).

Подробнее о свойствах нейронов см. [API-документацию Kaspersky Neuromorphic Platform](#). Если требуется, при [работе с исходным кодом](#) вы можете [реализовать и использовать собственные модели нейронов](#).

Библиотека synapse_traits

Библиотека `synapse_traits` представляет собой набор атрибутов [синапсов](#) ². Вы можете использовать синапсы, представленные в библиотеке, например, при построении проекций.

На текущий момент библиотека поддерживает наборы свойств для следующих синапсов:

- Синапс по модели [AdditiveSTDPDeltaSynapse](#).
- Синапс по модели [DeltaSynapse](#).
- Синапс по модели [SynapticResourceSTDPDeltaSynapse](#).

Подробнее о свойствах синапсов см. [API-документацию Kaspersky Neuromorphic Platform](#). Если требуется, при [работе с исходным кодом](#) вы можете [реализовать и использовать собственные модели синапсов](#).

Библиотека core

Библиотека `core` предоставляет поддержку бэкендов. Библиотека `core` предоставляет классы, необходимые для взаимодействия с бэкендом и выполнения прикладных задач.

Таблица ниже содержит описание компонентов библиотеки.

Компоненты библиотеки core

Компонент	Описание
<code>messaging</code>	Пространство имен, содержащее интерфейсы к сообщениям.
<code>tags</code>	Пространство имен, содержащее названия и типы значений известных тегов.
<code>Backend</code>	Базовый класс для конкретных реализаций бэкендов.
<code>BaseData</code>	Структура, определяющая набор базовых данных.
<code>continuously_uid_generator</code>	Класс, реализующий генератор уникальных идентификаторов.
<code>Device</code>	Базовый класс устройств.
<code>MessageBus</code>	Класс, реализующий интерфейс к шине сообщений.
<code>MessageEndpoint</code>	Класс, реализующий интерфейс к точке подключения к шине сообщений.
<code>Population</code>	Шаблонный класс, реализующий контейнер <code>нейронов</code> одной модели.
<code>Projection</code>	Шаблонный класс, реализующий контейнер <code>синапсов</code> одного типа.
<code>Subscription</code>	Класс, реализующий подписку на сообщения.
<code>TagMap</code>	Класс, реализующий интерфейс к словарю тегов.
<code>UID</code>	Класс, определяющий уникальные идентификаторы.
<code>uid_hash(.)</code>	Функция, реализующая хеширование UID.

На рисунке ниже представлена схема взаимодействия компонентов библиотеки.

В Kaspersky Neuromorphic Platform базовые данные `BaseData` содержат базовую информацию об объекте и включает в себя уникальный идентификатор `UID` и ассоциативный массив тегов `TagMap`. Базовые данные `BaseData` используются в проекции `Projection`, популяции `Population` и бэкенде `Backend`, соответственно каждый из них обладает идентификатором `UID` и набором тегов и их значениями `TagMap`.

Популяции `Population` и проекции `Projection` хранят данные, которые бэкенд обрабатывает и передает другим объектам платформы посредством сообщений, реализованных в классах пространства имен `messaging`. Сообщения содержат в своем заголовке `UID` отправителя.

Бэкенд Backend содержит устройство Device и шину сообщений MessageBus. Для подключения к шине сообщений MessageBus объекты платформы создают точки подключения MessageEndpoint, с которыми шина сообщений взаимодействует в процессе обмена сообщениями. Каждая точка подключения MessageEndpoint содержит контейнер подписок Subscription.

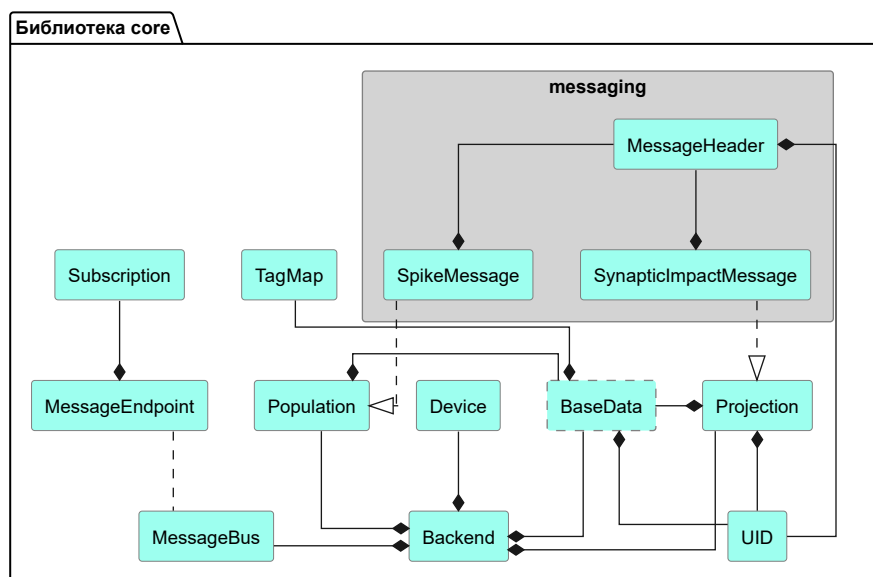


Схема взаимодействия компонентов библиотеки core

Пространство имен messaging

Пространство имен `messaging` реализует интерфейсы для доступа к сообщениям. 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.

Обмен сообщениями обеспечивается с помощью шины сообщений `MessageBus`. С помощью сообщений происходит обновление пользовательских структур и синхронизация изменений на бэкенде. Таблица ниже содержит описание поддерживаемых типов сообщений бэкенда.

Тип сообщения	Описание	Структура сообщения
SpikeMessage	Сообщение к проекции синапсов, содержащее спайк ^[?] . На сообщения, содержащие спайки, может быть подписано более одной проекции.	<code>header_</code> – заголовок сообщения, содержащий UID канала ввода или популяции, нейроны которой сгенерировали спайки, и время генерации спайков. <code>neuron_indexes_</code> – индексы нейронов популяции, сгенерировавших спайки. Индексы нейронов являются уникальными внутри популяции.
SynapticImpactMessage	Сообщение от проекции синапсов к популяции нейронов, содержащее синаптическое воздействие ^[?]	<code>header_</code> – заголовок сообщения, содержащий UID проекции и время отправки сообщения. <code>postsynaptic_population_uid_</code> – UID постсинаптической популяции. <code>presynaptic_population_uid_</code> – UID пресинаптической популяции. <code>is_forcing_</code> – флаг для включения и выключения пластичности в проекции синапсов. Если в проекции синапсов используется синаптическая пластичность, то атрибут имеет значение <code>true</code>, иначе имеет значение <code>false</code>. <code>impacts_</code> – вектор, содержащий следующие атрибуты: <code>connection_index_</code> – индекс соединения. <code>impact_value_</code> – значение синаптического воздействия. <code>synapse_type_</code> – тип синапса. <code>presynaptic_neuron_index_</code> – индекс пресинаптического нейрона, от которого был получен спайк синапсом проекции. Индекс нейрона является уникальным внутри популяции. <code>postsynaptic_neuron_index_</code> – индекс постсинаптического нейрона, которому будет передано значение синаптического воздействия для изменения значений его атрибутов. Индекс нейрона является уникальным внутри популяции. Вектор <code>impacts_</code> может содержать более одного набора перечисленных атрибутов для передачи синаптических воздействий нескольким нейронам одной популяции.

Пространство имен `messaging` предоставляет функции, с помощью которых вы можете получать сообщения из входного потока данных, отправлять сообщения в выходной поток данных, а также сравнивать сообщения между собой.

Kaspersky Neuromorphic Platform предоставляет методы и функции для взаимодействия с сообщениями. Например, с помощью метода отправки сообщений `MessageEndpoint::send_message()` вы можете отправить определенное сообщение через точку подключения к шине сообщений, а с помощью функций пространства имен `storage` вы можете загружать и сохранять сообщения `SpikeMessage` в [базовом формате KNP](#).

Подробнее о функциях пространства имен `messaging` и методах объектов платформы см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен tags

В пространстве имен `tags` реализованы названия и типы значений известных тегов.

Имена и типы значений известных тегов реализованы в пространстве имен `tags` для того, чтобы между бэкендом и фреймворком для C++ не возникало циркулярной зависимости.

Класс Backend

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

- взаимодействовать с физическими устройствами (например, получать информацию о потребляемой мощности);
- управлять обучением и исполнением [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.

Перед инициализацией бэкенда вы можете загрузить проекции, содержащие [синапсы](#) и популяции, содержащие [нейроны](#). Для этого вы можете использовать методы `load_all_projections()` и `load_all_populations()` базового класса `Backend` или соответствующие методы конкретного класса бэкенда. Если предварительно была создана модель `Model`, в которую была передана нейронная сеть `Network`, хранящая популяции `Population` и проекции `Projection`, вы можете использовать метод загрузки модели `ModelLoader::load()` для ее загрузки на бэкенд.

Конкретные классы бэкендов являются наследниками базового класса `Backend`. Для инициализации конкретного бэкенда необходимо переопределить его методы в классе-наследнике. В процессе инициализации загруженные проекции синапсов подписываются на [спайки](#) от связанных популяций или каналов ввода, а загруженные популяции подписываются на [синаптические воздействия](#) от связанных проекций. При необходимости вы можете подписать необходимые объекты платформы на сообщения с помощью метода подписки `subscribe()`.

Для выбора устройства, на котором будет исполняться нейронная сеть, необходимо сначала вызвать метод `get_devices()`, который предоставляет вычислители конкретного типа. Тип вычислителей является наследником класса `Device`. Наследники определены в пространстве имен `devices` и реализуют конкретные устройства. Каждое устройство обладает своим набором функций и свойств. После этого необходимо вызвать метод `select_devices()` или `select_device()` для выбора конкретного устройства из предоставленных, передав в него уникальные идентификаторы устройств или указатели на экземпляры класса `Device`.

Базовый класс `Backend` также предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- Запускать и останавливать обучение нейронной сети.
- Включать и выключать использование синаптической пластичности во время обучения нейронной сети.

- Запускать и останавливать исполнение нейронной сети.
- Получать следующие данные:
 - [Базовые данные](#) бэкенда.
 - [Шину сообщений](#). С помощью шины сообщений бэкенд обеспечивает обмен сообщениями между объектами платформы, а также обмен сообщениями от [модификаторов](#) к [нейронам](#) и [синапсам](#).
 - [Точку подключения](#) к шине сообщений.
 - Статус исполнения нейронной сети на бэкенде.
 - Устройства, поддерживаемые бэкендом.
 - Номер шага исполнения нейронной сети.

Подробнее о методах базового класса `Backend` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Структура BaseData

Структура `BaseData` определяет набор базовых данных, общих для некоторых объектов платформы. Например, базовые данные `BaseData` используются в объектах классов `Backend`, `Device`, `Population` и `Projection`.

Базовые данные `BaseData` включают в себя следующие атрибуты:

`uid`

Уникальный идентификатор объекта.

`tags`

Ассоциативный массив `TagMap`, состоящий из тегов, используемых объектом, и их значений.

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.

Подробнее о методах класса `continuously_uid_generator` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Класс Device

Класс `Device` является базовым для классов 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.

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

Подробнее о базовом классе `Device` и его методах см. в [API-документации Kaspersky Neuromorphic Platform](#).

Класс MessageBus

Класс `MessageBus` реализует интерфейс для доступа к шине сообщений. Шина сообщений обеспечивает прием, распределение и доставку [сообщений](#) между подключенными к ней объектами.

Класс `MessageBus` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- создавать шины сообщений;
- создавать точки подключения к шине сообщений;
- принимать, распределять доставлять сообщения между объектами, подключенными к шине сообщений;
- получать используемые точки подключения.

Шина сообщений предоставляет множество точек подключения [MessageEndpoint](#), с помощью которых объекты платформы и объекты [внешней среды](#) могут подписаться на сообщения или отправить их. Вы можете создать точки подключения к шине сообщений и затем подписать созданные точки подключения на сообщения определенного типа с помощью метода подписки [MessageEndpoint::subscribe\(\)](#). 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 [Subscription](#) фильтрует сообщения и отправляет их подписчикам. Подобный подход позволяет избежать создания большого количества IPC-каналов, поскольку издатель и подписчики не связаны напрямую.

Подробнее о методах шины сообщений `MessageBus` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Класс MessageEndpoint

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

Вы можете создать точку подключения следующими способами:

- с помощью метода создания точки подключения [MessageBus::create_endpoint\(\)](#);
- с помощью конструктора перемещения [MessageEndpoint](#).

Точка подключения может хранить в себе набор подписок [Subscription](#) на сообщения от различных отправителей и для различных получателей, а также набор уникальных идентификаторов отправителей, на которые она подписана. Подписки используются для фильтрации и хранения сообщений, поступивших в шину сообщений.

Класс `MessageEndpoint` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- Подписывать точку подключения на сообщения определенного типа.
- Удалять подписки на сообщения от определенных отправителей.
Если вы удалите точку подключения, то связанные объекты подписки также будут удалены. Передача сообщений с помощью этой точки подключения прекратится.
- Получать сообщения от отправителя, на который подписана точка подключения.
Перед вызовом метода необходимо использовать метод маршрутизации сообщений [MessageBus::route_messages\(\)](#) для получения сообщений точкой подключения.
- Отправлять сообщения с помощью шины сообщений.
- Получать подписки, связанные с точкой подключения.
- Получать набор уникальных идентификаторов отправителей, на которые подписана точка подключения.
- Получать уникальный идентификатор получателя сообщений по подписке.

Подробнее о методах точки подключения `MessageEndpoint` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Класс Population

Шаблонный класс `Population` реализует контейнер, который содержит [нейроны](#) принадлежащие одной модели. При создании объекта популяции необходимо указать количество нейронов и генератор нейронов определенного типа. Вы можете указать один из типов нейронов, поддерживаемых библиотекой [neuron_traits](#).

Вы можете передать данные популяций на бэкенд и вычислить их атрибуты с помощью методов бэкенда. Для передачи данных популяции на бэкенд вы можете использовать метод загрузки популяций [Backend::load_all_populations\(\)](#). После вычисления новые атрибуты популяций хранятся на бэкенде. Для получения от бэкенда текущих параметров нейронной сети вы можете вызвать синхронизацию. Если вы вручную изменили атрибуты популяции до вызова синхронизации, вы можете заново передать данные популяции на бэкенд для вычисления новых атрибутов.

Класс `Population` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- добавлять нейроны в ранее созданную популяцию;
- удалять один или несколько нейронов из популяции;
- получать базовые данные популяции;
- получать атрибуты нейронов популяции;
- получать информацию о количестве нейронов в популяции.

Подробнее о методах [популяций](#) и [бэкендов](#) см. в API-документации Kaspersky Neuromorphic Platform.

Класс Projection

Шаблонный класс `Projection` реализует контейнер, который содержит [синапсы](#) ² одного из типов, поддерживаемых библиотекой [synapse traits](#). При создании объекта проекции необходимо указать уникальные идентификаторы пресинаптической и постсинаптической популяций, которые необходимо связать этой проекцией.

Проекция представляет собой набор соединений нейронов пресинаптической и постсинаптической популяций [Population](#). [Обмен данными](#) между популяциями нейронов и проекциями синапсов осуществляется через [сообщения](#). Проекция синапсов получает сообщения [SpikeMessage](#) со [спайками](#) ² от пресинаптической популяции и отправляет сообщения [SynapticImpactMessage](#) с [синаптическими воздействиями](#) ² в постсинаптическую популяцию.

С помощью метода загрузки проекций [Backend::load_all_projections\(\)](#) вы можете загрузить данные созданной проекции на бэкенд для вычисления ее атрибутов.

Класс `Projection` также предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- Добавлять синапсы в проекцию.
- Удалять синапсы из проекции. Вы можете удалять синапсы в соответствии с некоторым заданным критерием или синапсы, связанные с определенным пресинаптическим или постсинаптическим нейроном.
- Получать базовые данные проекции.
- Получать уникальные идентификатор популяций, связанных с проекцией.
- Получать атрибуты синапсов проекции.

- Получать информацию о количестве синапсов проекции.
- Блокировать и разблокировать возможность изменения весов синапсов.

Каждая проекция синапсов содержит следующие атрибуты:

- 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.
- Атрибуты, используемые синапсами одного типа.

- Реестр соединений синапсов проекции с пресинаптическими и постсинаптическими нейронами.

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.

- Значение типа `boolean` для обозначения актуальности реестра соединений. Если реестр соединений актуален, то атрибут имеет значение `true`, иначе имеет значение `false`.

Подробнее о методах проекции см. в [API-документации Kaspersky Neuromorphic Platform](#).

Класс Subscription

Класс `Subscription` реализует функциональность подписки на сообщения и хранит в себе поступившие [сообщения](#). Вы можете создать подписку одним из следующих способов:

- с помощью конструктора [Subscription](#);
- с помощью метода подписки [MessageEndpoint::subscribe\(\)](#).

Подписка обеспечивает [фильтрацию сообщений](#). Фильтрация сообщений осуществляется в соответствии с уникальными идентификаторами отправителей и получателей, определенных в объекте подписки. Сообщения хранятся в подписке до их запроса получателем или до их удаления из подписки.

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

Класс `Subscription` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- добавлять отправителей сообщений в подписку;
- удалять отправителей сообщений из подписки;

- добавлять сообщения в подписку;
- удалять сообщения, хранящиеся в подписке;
- получать уникальные идентификаторы отправителей и получателя сообщений;
- получать сообщения, хранящиеся в подписке;
- проверять, связана ли подписка с отправителем с заданным уникальным идентификатором.

Подробнее о методах класса `Subscription` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Класс TagMap

Класс `TagMap` 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. Ассоциативный массив тегов является составляющей частью базовых данных `BaseData` объектов платформы.

Класс `TagMap` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- Получать значение тега, передав в метод его имя.
- Проверять наличие записей в ассоциативном массиве.

Подробнее о методах класса `TagMap` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Класс UID

Класс `UID` реализует уникальный идентификатор объекта. Идентификатор создается с помощью конструктора `UID` и является частью базовых данных `BaseData` объектов платформы.

Класс `UID` предоставляет операторы, с помощью которых вы можете сравнивать идентификаторы объектов, а также проверять идентификаторы на корректность. Если требуется, вы можете получить хеш-значение идентификатора объекта с помощью функции `uid hash()`.

Подробнее о конструкторах и операторах класса `UID` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Функция `uid_hash()`

Функция `uid_hash()` реализует функцию хеширования идентификаторов объектов.

Библиотека `meta`

Библиотека `meta` представляет собой набор шаблонов общего назначения. Шаблоны библиотеки используются, например, внутри функций бэкенда, обеспечивающих интроспекцию.

Пространство имен `devices`

Пространство имен `devices` предоставляет набор интерфейсов, которые вы можете использовать для взаимодействия с физическими устройствами.

Функции и классы, реализованные в пространстве имен, используются в бэкендах для получения информации об устройствах. Для выбора устройства, на котором будет исполняться нейронная сеть, необходимо сначала вызвать метод `Backend::get_devices()`, который предоставляет типы вычислителей. Типы вычислителей являются наследниками класса `Device`. Наследники определены в пространстве имен `devices` и реализуют конкретные устройства. Каждое устройство обладает своим набором функций и свойств. Для выбора конкретного устройства из предоставленных необходимо вызвать метод `select_devices()` или `select_device()` базового класса `Backend`, передав в него уникальные идентификаторы устройств или указатели на экземпляры класса `Device`.

Информацию об устройстве (например, о потребляемой мощности) вы можете получить с помощью методов объекта, который обеспечивает интерфейс к устройству.

Если требуется, при [работе с исходным кодом](#) вы можете реализовать и использовать собственные классы для взаимодействия с устройствами. Для этого в рамках бэкенда реализуйте класс устройства и предоставьте его интерфейс, а также реализуйте метод загрузки устройства, которая возвращает указатель на экземпляр класса `Device`.

В Kaspersky Neuromorphic Platform на текущий момент реализованы интерфейсы для центрального процессора, графического процессора с поддержкой технологии CUDA и нейроморфного процессора Алтай-2. При использовании нейроморфного процессора Алтай-2 вы можете выбрать в качестве типа устройства как аппаратный процессор, так и его программный эмулятор. Для аппаратного процессора используется фиксированная конфигурация ядер. При выборе программного эмулятора в качестве типа устройства вы можете [изменить параметры его конфигурации](#).

Подробнее о функциях пространства имен `devices` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Фреймворк для C++

Фреймворк для C++ предназначен для упрощения процессов создания [импульсных нейронных сетей](#) и их обучения. Полученные в результате нейронные сети могут исполняться на центральном процессоре, [нейроморфном процессоре Алтай-2](#) или его программном эмуляторе.

Фреймворк для C++ представляет собой независимые блоки, которые обеспечивают управление бэкендами, преобразуют представление нейронной сети в разные форматы и выбирают оптимальную нейронную сеть для запуска. Во фреймворке для C++ реализованы функции, не относящиеся непосредственно к обучению и исполнению 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 бэкенды.
- Вводить и выводить данные нейронных сетей.
- Контролировать выполнение обучения и исполнения импульсных нейронных сетей.
- Реализовывать специальные алгоритмы.

Фреймворк для C++ реализован в пространстве имен [framework](#). Таблица ниже содержит описание компонентов фреймворка для C++.

Компоненты фреймворка для C++

Компонент		Описание
coordinates		Пространство имен, которое содержит пространственные координаты нейрона.
data_processing		Пространство имен, которое предоставляет функциональность для обработки данных и обучения моделей классификации, а также для обработки изображений для задач классификации.
inference_evaluation		Пространство имен, которое предоставляет функциональность для оценки качества работы модели классификации и записи результатов в удобном формате.
io		Пространство имен, которое содержит пространства имен input , output и storage .
	input	Пространство имен, которое содержит интерфейсы для доступа к каналу ввода и преобразователю данных, поступающих из внешней среды .
	output	Пространство имен, которое содержит интерфейсы для доступа к каналам вывода.
	storage	Пространство имен, которое содержит функции загрузки и сохранения сообщений SpikeMessage .
modifier		Пространство имен, которое содержит интерфейсы к обработчикам сообщений.
monitoring		Пространство имен, которое содержит интерфейс к наблюдателю и функции для мониторинга работы модели.
normalization		Пространство имен, которое содержит функции и функторы для нормализации параметров синапсов и нейронов в нейронных сетях.
population		Пространство имен, которое содержит функции для генерации популяций нейронов.
projection		Пространство имен, которое содержит функции для генерации проекций синапсов и генерации синапсов со значениями атрибутов по умолчанию.
sonata		Пространство имен, которое содержит функции загрузки и сохранения нейронной сети в формате SONATA .
synchronization		Пространство имен, которое содержит функцию для получения актуального состояния нейронной сети с бэкенда.

Компонент	Описание
BackendLoader	Класс, реализующий загрузчик динамических библиотек.
Model	Класс, реализующий модель.
ModelExecutor	Класс, реализующий исполнитель модели.
ModelLoader	Класс, реализующий загрузчик модели.
Network	Класс, реализующий объект нейронной сети.

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

В Kaspersky Neuromorphic Platform нейронная сеть `Network` может быть [загружена](#) в формате `SONATA` или создана вручную. Созданную нейронную сеть вы можете наполнить проекциями синапсов [Projection](#) и популяциями нейронов [Population](#). Вы можете сгенерировать проекции и популяции с помощью функций пространств имен `projection` и `population` соответственно. При необходимости вы можете нормализовать значения атрибутов нейронов популяций и синапсов проекций с помощью функций пространства имен `normalization`.

Вы можете нарисовать нейронную сеть `Network` с помощью набора координат (произвольных или задаваемых пользователем) из пространства имен `coordinates`. Состояние нейронной сети `Network` может быть актуализировано с бэкенда [Backend](#) с помощью функции пространства имен `synchronization`.

Нейронная сеть `Network`, а также контейнеры канала ввода [InputChannel](#) и канала вывода [OutputChannel](#), реализованных в пространствах имен `input` и `output` соответственно, хранятся в модели `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 модель `Model` с помощью метода исполнения моделей `ModelExecutor::start()`. Исполнитель моделей `ModelExecutor` также содержит наблюдателя `MessageObserver` из пространства имен `monitoring`, обработчика сообщений из пространства имен `modifier` и загрузчик моделей `ModelLoader`, с помощью методов которого вы можете загружать модель на конкретный экземпляр бэкенда. Для обработки данных модели вы можете использовать функции пространства имен `data_processing`. Для исполнения модели на экземпляре бэкенда исполнителю моделей `ModelExecutor` передается указатель на экземпляр бэкенда, полученный от загрузчика бэкендов `BackendLoader`. После исполнения модели вы можете оценить качество работы модели с помощью функций пространства имен `inference_evaluation`.

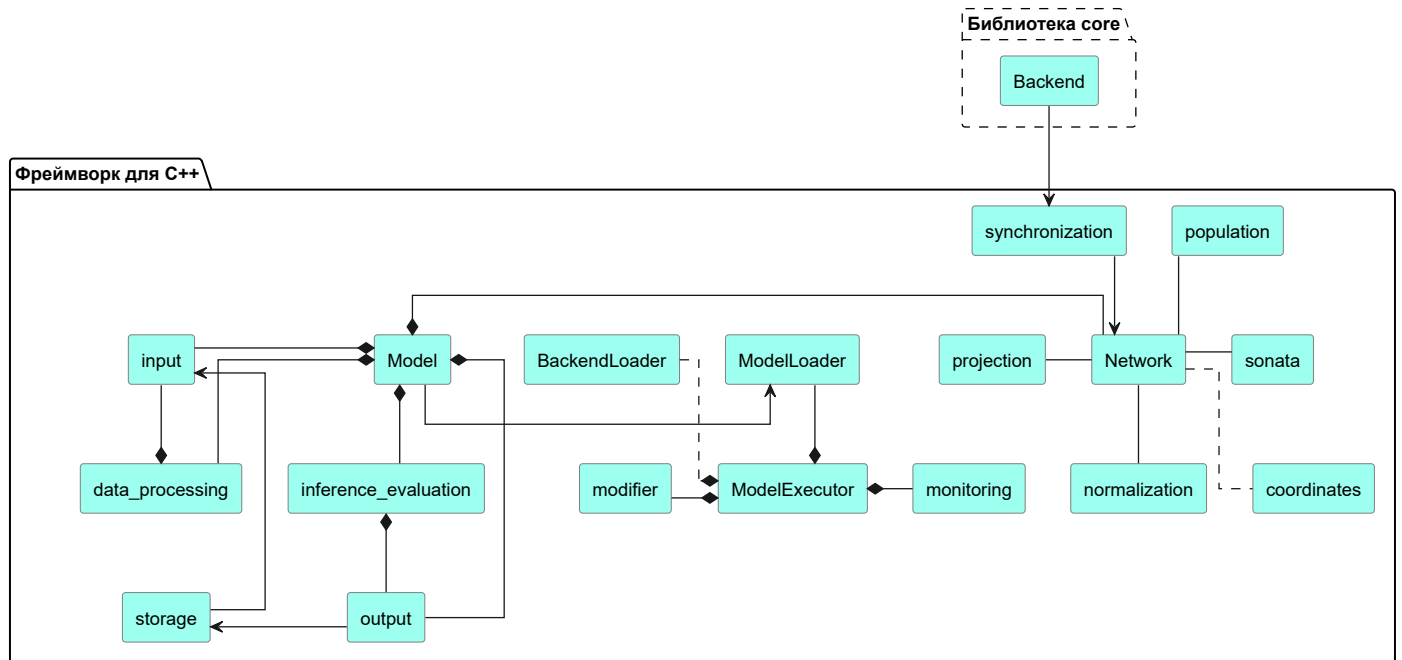


Схема взаимодействия компонентов фреймворка для C++

Пространство имен coordinates

В пространстве имен `coordinates` реализованы 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 система координат для двухмерного и трехмерного пространства, а также полярная и сферическая системы координат.

Если требуется, вы можете реализовать и использовать собственные системы координат.

Подробнее о пространстве имен `coordinates` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен data_processing

Пространство имен `data_processing` предоставляет функциональность для обработки данных для обучения моделей классификации, а также для обработки изображений для задач классификации.

Базовый класс Dataset

Пространство имен `classification` в составе пространства имен `data_processing` содержит класс `Dataset`, который является базовым для классов наборов данных, которые вы можете использовать для обучения и тестирования моделей классификации.

Базовый класс `Dataset` предоставляет следующие методы:

`split()`

Разделяет набор данных на обучающую и тестовые выборки. Обучающие и тестовые выборки представляют собой вектор пар меток классов объектов и векторов бинарных значений, соответствующих наличию или отсутствию спайка. В свою очередь наличие спайка соответствует принадлежности объекта определенному классу.

`get data for training()`

Получает данные для обучения модели классификации.

`get data for inference()`

Получает данные для инференса модели классификации.

`get_steps_per_frame()`

Получает количество шагов, необходимых для обработки одного класса объектов.

`get_steps_required_for_training()`

Получает количество шагов, необходимых для обучения модели классификации.

`get_steps_required_for_inference()`

Получает количество шагов, необходимых для инференса модели классификации.

`get_required_training_amount()`

Получает информацию о количестве объектов, используемых для обучения модели классификации.

`get_amount_of_classes()`

Получает информацию о количестве классов в наборе данных.

Пространство имен images

Пространство имен `classification` также содержит пространство имен `images`, используемое для получения сведений о классификации изображений. В свою очередь пространство имен `images` содержит класс `Dataset`, который является наследником базового класса `Dataset` и предоставляет дополнительные методы для обработки изображений.

Класс `Dataset` в пространстве имен `images` предоставляет следующие методы:

`process_labels_and_images()`

Считывает поток изображений и их меток классов и формирует набор данных для обучения и инференса модели классификации изображений путем преобразования изображений в спайки.

`make_training_labels_generator()`

Создает генератор набора меток, соответствующих классам изображений, для обработки моделью за шаг ее исполнения.

`make_training_images_spikes_generator()`

Создает генератор набора изображений в виде спайков для обработки моделью за шаг ее исполнения во время ее обучения.

`make_inference_images_spikes_generator()`

Создает генератор набора изображений в виде спайков для обработки моделью за шаг ее исполнения во время ее инференса.

`make_incrementing_image_to_spikes_converter()`

Создает преобразователь, который преобразует изображение в виде вектора байтов в вектор спайков.

`get_image_size()`

Получает размер изображения в байтах.

Подробнее о классах пространства имен `data_processing` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен `inference_evaluation`

Пространство имен `inference_evaluation` предоставляет функциональность для оценки качества работы модели и записи результатов в формате CSV.

Функции пространства имен `inference_evaluation`

В пространстве имен `inference_evaluation` реализованы функции для получения метрик оценки модели:

`get_precision()`

Получает метрику *точность* (англ. *precision*), показывающую, какая доля положительных результатов на самом деле являются положительными.

`get_recall()`

Получает метрику *полнота* (англ. *recall*), показывающую, какую долю объектов класса модель правильно классифицировала из всех объектов этого класса.

`get_prevalence()`

Получает метрику *распространенность* (англ. *prevalence*), показывающую, долю объектов с положительными состояниями.

`get_accuracy()`

Получает метрику *правильность* (англ. *accuracy*), показывающую, долю правильно предсказанных классов ко всем предсказаниям.

`get_f_score()`

Получает метрику *F-мера* (англ. *F-score*), показывающую гармоническое среднее между точностью и полнотой.

Формулы для расчета метрик оценки качества модели см. в [Приложении](#).

Пространство имен classification

Пространство имен `inference_evaluation` содержит пространство имен `classification`. В пространстве имен реализован класс `InferenceResult`, который предоставляет интерфейс для работы с оценками модели по результатам классификации для одного класса объектов. С помощью класса `InferenceResult` вы можете получить информацию о результатах классификации и общем количестве голосов. Общее количество голосов представляет собой сумму истинно положительных, ложно положительных и ложно отрицательных результатов.

В пространстве имен `classification` также реализован класс `InferenceResultsProcessor`, который обрабатывает результаты оценки модели классификации для всех классов объектов. В классе реализованы следующие методы:

`process_inference_results()`

Обрабатывает результаты оценок для всех классов объектов.

`get_inference_results()`

Получает результаты оценки модели для каждого класса объектов.

`write_inference_results_to_stream_as_csv()`

Записывает результаты оценки модели классификации в поток в формате CSV. Описание данных, передаваемых в CSV-формате, см. в [Приложении](#).

Подробнее о классах и функциях пространства имен `inference_evaluation` см. в [API-документации Kaspersky Neuromorphic Platform](#).

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. При создании преобразователя данных необходимо указать поток данных `istream`, из которого поступают данные, функцию преобразования входных данных и длину отрезка входных данных. При необходимости вы можете изменить длину отрезка входных данных.

Преобразователь данных `SequenceConverter` возвращает список индексов входных значений, которые содержат спайки, в виде сообщения `SpikeMessage`. Если требуется, вы можете реализовать и использовать собственные функции преобразования входных данных.

Класс `SequenceConverter` предоставляет методы, с помощью которых вы можете получить поток данных, используемый преобразователем данных, а также вызвать функцию преобразования данных из потока `istream` в сообщение `SpikeMessage`.

Класс `IndexConverter`

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

При создании преобразователя данных необходимо указать поток данных `istream`, из которого поступают данные, и символ, с помощью которого индексы нейронов, сгенерировавших спайки, будут разделены в сообщении `SpikeMessage`.

С помощью методов класса `IndexConverter` вы можете получить поток данных, используемый преобразователем данных, а также вызвать функцию преобразования данных из потока `istream` в сообщение `SpikeMessage`.

Класс `InputChannel`

Класс `InputChannel` реализует канал ввода, который принимает сообщения `SpikeMessage` от преобразователя данных. При создании канала ввода необходимо указать уникальный идентификатор UID канала ввода, точку подключения `MessageEndpoint`, а также функтор для преобразования данных в сообщения `SpikeMessage`.

После создания канал ввода необходимо связать с точкой подключения и получателем сообщений с помощью метода `input::connect input()`. При наличии подписки `Subscription` данные, преобразованные в сообщения `SpikeMessage`, могут быть переданы через точки подключения с помощью методов класса `InputChannel`.

Если вы загрузили модель `Model`, хранящую нейронную сеть `Network`, каналы ввода и вывода, в исполнитель моделей `ModelExecutor`, то при ее исполнении на бэкенде `Backend` автоматически осуществляется передача отправленных сообщений до точки подключения через шину сообщений.

Класс `InputChannel` также предоставляет методы, с помощью которых вы можете получить базовые данные канала ввода.

Каждый канал ввода содержит следующие атрибуты:

- Точку подключения к шине сообщений, с помощью которой преобразованные данные отправляются подписчикам.
- Преобразователь данных в сообщения `SpikeMessage`.

Подробнее о функциях и интерфейсах пространства имен `input` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен modifier

Пространство имен `modifier` содержит классы, реализующие модификаторы.

`KWtaRandomHandler`

Предоставляет обработчик сообщений, который получает вектор сообщений `SpikeMessage` и возвращает определенное количество случайных индексов нейронов, сгенерировавших спайки.

`GroupWtaRandomHandler`

Предоставляет обработчик сообщений, который получает вектор сообщений `SpikeMessage` и разделяет их на группы. В каждой группе обработчик сортирует сообщения по количеству нейронов, сгенерировавших спайки, в убывающем порядке и возвращает индексы нейронов, сгенерировавших спайки, в определенном количестве отсортированных сообщений группы.

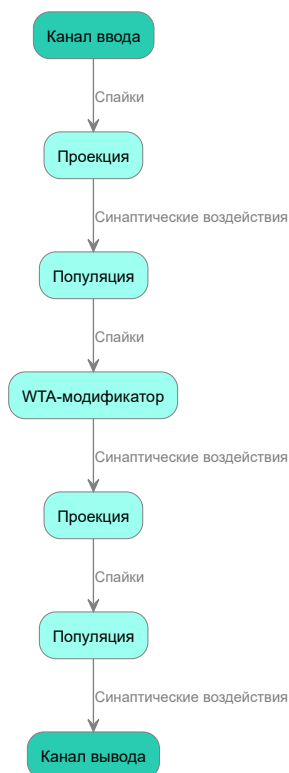
`KWtaPerGroup`

Предоставляет обработчик сообщений, который получает вектор сообщений `SpikeMessage` и разделяет их на группы. Для каждой группы сообщений обработчик возвращает определенное количество случайных индексов нейронов, сгенерировавших спайки.

`SpikeUnionHandler`

Предоставляет обработчик сообщений, который получает вектор сообщений `SpikeMessage` и возвращает индексы нейронов, сгенерировавших спайки, хотя бы в одном сообщении.

Классы `KWtaRandomHandler`, `GroupWtaRandomHandler` и `KWtaPerGroup` реализуют [механизм Winner Takes ALL \(WTA\)](#)². Каждый из этих модификаторов принимает сообщения `SpikeMessage`, содержащие спайки, и отправляет сообщения `SynapticImpactMessage`, содержащие синаптические воздействия. В силу этого при построении нейронной сети нейронный слой с WTA-модификатором необходимо подключить к популяции нейронов на входе и проекции синапсов на выходе (см. рис. ниже).



Структура нейронной сети с WTA-модификатором

Вы можете добавить обработчик сообщений в исполнитель моделей с помощью метода `ModelExecutor::add_observer()` или функции `projection::add_wta_handlers()`.

Подробнее об интерфейсах пространства имен `modifier` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен monitoring

Пространство имен `monitoring` содержит класс `MessageObserver` и пространство имен `model`.

Класс MessageObserver

Класс `MessageObserver` реализует наблюдателя. При создании наблюдателя необходимо указать следующие параметры:

- Точку подключения `MessageEndpoint` к шине сообщений, которая обеспечивает подписку на сообщения сущностей;
- Функцию для обработки полученных сообщений;
- Идентификатор `UID` наблюдателя.

С помощью методов класса `MessageObserver` вы можете подписать наблюдателя на некоторый набор сущностей и определенный тип сообщений. В конце каждого такта исполнения нейронной сети `Network` наблюдатель получает сообщения по подписке `Subscription` и обрабатывает их в соответствии с заданной функцией.

Пространство имен model

В пространстве имен `model` реализованы функции для мониторинга работы модели:

`add_aggregated_spikes_logger()`

Добавляет к модели наблюдателя, который с заданной периодичностью собирает информацию о спайках от указанных отправителей и записывает в поток вывода в табличном виде полученную информацию. Каждая строка таблицы является последовательностью из чисел, соответствующих номеру шага исполнения модели, на котором был сгенерирован спайк и количеству спайков для каждого отправителя.

Пример записи функции `add_aggregated_spikes_logger()`

```
Index, Population1, Population2, Population3
0, 5, 3, 8
15, 2, 7, 1
17, 15, 4, 9
```

`add_projection_weights_logger()`

Добавляет к модели наблюдателя, который с заданной периодичностью собирает информацию о весах синапсов в указанной проекции и записывает в поток вывода в структурированном формате информацию о номере шага исполнения модели, индексе нейрона, который сгенерировал спайк, текущих значениях весов и последних моментах их обновления, разделенных символом `|`.

Пример записи функции `add_projection_weights_logger()`

```
Step: 1
Neuron 1
0.123|50 0.456|45 0.789|40
```

`add_spikes_logger()`

Добавляет к модели наблюдателя, который собирает информацию о спайках в сообщениях `SpikeMessage` от указанных отправителей и для каждого спайка записывает в поток вывода построчно информацию о номере шага исполнения модели, на котором были сгенерированы спайки, имени отправителя спайков и индексы нейронов, которые сгенерировали спайки.

Пример записи функции `add_spikes_logger()`

```
Step: 1
Sender: Population1
15 23 42 57 89
Step: 2
Sender: Population1
5
```

`add_status_logger()`

Добавляет к модели наблюдателя, который с заданной периодичностью отслеживает сообщения `SpikeMessage` от всех популяций нейронов в модели и записывает в поток вывода информацию о количестве полученных сообщений за шаг исполнения модели.

Подробнее о пространстве имен `monitoring` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен `normalization`

Пространство имен `normalization` предоставляет функции и функторы для нормализации параметров синапсов и нейронов в нейронных сетях `Network`. Эти функции и функторы обеспечивают устойчивую работу модели `Model` и предотвращают проблемы, связанные с несбалансированными данными.

Функции пространства имен `normalization`

`normalize_synapses()`

Нормализует атрибуты синапсов в конкретной проекции `Projection` или в каждой проекции нужной нейронной сети `Network`. Функция итеративно обрабатывает каждый синапс в проекции. Для каждого синапса функция получает значение нужного атрибута, корректирует его значение в соответствии с заданной функцией и устанавливает для атрибута скорректированное значение.

`normalize_neurons()`

Нормализует атрибуты нейронов в конкретной популяции `Population` или в каждой популяции нужной нейронной сети `Network`. Функция итеративно обрабатывает каждый нейрон в популяции. Для каждого нейрона функция получает значение нужного атрибута, корректирует его значение в соответствии с заданной функцией и устанавливает для атрибута скорректированное значение.

Функции пространства имен `normalization` являются шаблонными, поэтому вы можете использовать их при работе с любым типом нейронов и синапсов для нормализации значений нужных атрибутов.

Функторы пространства имен `normalization`

`PotentialAccessor`

Читает и обновляет атрибут `potential_` при нормализации нейронов. В атрибуте хранится значение потенциала мембраны нейрона. Функтор является шаблонным, поэтому вы можете использовать его для любых типов нейронов.

`WeightAccessor`

Читает и обновляет атрибут `weight_` при нормализации синапсов. В атрибуте хранится значение веса синапса. Функтор является шаблонным, поэтому вы можете использовать его для любых типов синапсов.

`Rescaler`

Линейно масштабирует значения атрибутов из одного диапазона в другой. Функтор является шаблонным и поддерживает любой числовой тип данных.

При необходимости вы можете реализовать и использовать собственные функторы для доступа к атрибутам нейронов и/или синапсов, а также для нормализации их значений.

Подробнее о функциях и функторах пространства имен `normalization` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен `output`

В пространстве имен `output` реализован интерфейс для доступа к каналу вывода и преобразователи сообщений `SpikeMessage`. Сообщения `SpikeMessage` содержат спайки. 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 сообщения со спайками от связанной популяции `Population`. Сообщения со спайками передаются в канал вывод за несколько тактов исполнения нейронной сети. При создании канала вывода необходимо указать уникальный идентификатор `UID` канала вывода и точку подключения `MessageEndpoint`.

Сообщения `SpikeMessage`, полученные каналом вывода, могут быть преобразованы в один из следующих форматов данных:

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

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

Преобразователи сообщений `SpikeMessage`

Пространство имен `output` предоставляет следующие объекты для преобразования полученных сообщений `SpikeMessage`:

Класс `ConvertToSet`

Реализует преобразователь данных, который получает сообщения `SpikeMessage` и преобразует их в вектор индексов нейронов, сгенерировавших спайки.

Функтор `converter bitwise`

Реализует преобразователь данных, который получает сообщения `SpikeMessage` и преобразует их в вектор значений типа `boolean`, в котором значение `true` соответствуют наличию спайка.

Функтор `converter count`

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

Подробнее о функциях и интерфейсах пространства имен `output` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен `population`

Пространство имен `population` содержит пространства имен `creators` и `neurons_generators`. Пространства имен предоставляют функции и интерфейсы, необходимые для генерации популяций и их наполнения нейронами определенного типа.

При необходимости для создания популяций и нейронов вы можете использовать конструктор класса [Population](#).

Пространство имен creators

В пространстве имен `creators` реализованы функции для генерации популяций:

`from_container()`

Генерирует популяции из контейнера, содержащего атрибуты нейронов.

`make_random()`

Генерирует популяции со случайными значениями атрибутов нейронов.

`make_default()`

Генерирует популяции со значениями атрибутов нейронов по умолчанию.

`make_clone()`

Генерирует популяции, значения атрибутов нейронов которых скопированы с атрибутов заданного нейрона.

Пространство имен neurons_generators

В пространстве имен `neurons_generators` реализованы функции и интерфейсы для генерации нейронов:

Функция `from_container()`

Генерирует нейроны из контейнера, содержащего атрибуты нейронов.

Конструктор `MakeRandom()`

Генерирует нейроны со случайными значениями атрибутов.

Функция `make_default()`

Генерирует нейроны со значениями атрибутов по умолчанию.

Функция `make_clone()`

Генерирует нейроны, значения атрибутов которых скопированы с атрибутов заданного нейрона.

Подробнее о функциях и интерфейсах пространства имен `population` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен projection

Пространство имен `projection` содержит пространства имен `creators`, `synapse_generators` и `parameters_generators`. Пространства имен предоставляют функции и интерфейсы, необходимые для генерации проекций и их наполнения синапсами определенного типа. Перед вызовом функций необходимо предварительно создать популяции, которые необходимо связать проекцией.

В пространстве имен также реализованы функции, с помощью которых вы можете создать новую проекцию, связав две популяции одним из следующих образов:

- случайным образом;
- полностью (каждый нейрон пресинаптической популяции связан с каждым нейроном постсинаптической популяции).

С помощью функций пространства имен вы также можете добавить в исполнитель моделей `ModelExecutor` обработчик сообщений `SpikeMessage` из пространства имен `modifier`, реализующий механизм `Winner Takes ALL (WTA)` [\[2\]](#).

При необходимости для создания проекций и синапсов вы можете использовать методы, предоставляемые классом `Projection`.

Пространство имен `creators`

В пространство имен `creators` реализованы функции для генерации проекций синапсов. Эти функции позволяют генерировать проекции синапсов путем следующих соединений нейронов:

- Полное соединение – каждый нейрон пресинаптической популяции соединяется с каждым нейроном постсинаптической популяции.
- Последовательное соединение – нейроны популяции меньшего размера соединяются с нейронами другой популяции в определенном порядке.
- Исключающее соединение – нейроны популяций не соединяются, если их индексы совпадают.
- Один-к-одному – нейроны пресинаптической и постсинаптической популяций соединяются в отношении один-к-одному.
- Случайное соединение – нейроны популяций соединяются с определенной вероятностью.
- Случайное соединение с ограничением – каждый нейрон одной популяции соединяется с определенным количеством случайных нейронов другой популяции.
- Условное соединение – нейроны соединяются, если функция генерации синапса возвращает ненулевое `std::optional` значение.

Вы также можете сгенерировать проекции синапсов следующими способами:

- Из контейнера, содержащего атрибуты синапсов и индексы пресинаптического и постсинаптического нейронов.
- Из объекта `std::map`, где ключами являются индексы пресинаптического и постсинаптического нейронов, а значениями – атрибуты синапсов.
- Путем копирования синапсов из другой проекции.

Пространство имен `synapse_generators`

В пространстве имен `synapse_generators` реализованы функции и интерфейсы, с помощью которых вы можете генерировать синапсы, соединяя нейроны следующими способами:

- Полное соединение – каждый нейрон пресинаптической популяции соединяется с каждым нейроном постсинаптической популяции.
- Последовательное соединения – нейроны популяции меньшего размера соединяются последовательно с нейронами другой популяции.

- Исключающее соединение – каждый нейрон одной популяции соединяется с каждым нейроном другой популяции, за исключением тех, чьи индексы совпадают.
- Один-к-одному – нейроны популяций соединяются в отношении один-к-одному.
- Из контейнера, содержащего атрибуты синапсов и индексы пресинаптического и постсинаптического нейронов.
- Из объекта `std::map` или `std::unordered_map`, содержащего атрибуты синапсов в качестве значения и индексы пресинаптического и постсинаптического нейронов в качестве ключа.
- Случайное соединение – каждый нейрон пресинаптической популяции соединяется с каждым нейроном постсинаптической популяции с определенной вероятностью.
- Случайное соединение с ограничением – каждый нейрон одной популяции соединяется с определенным количеством случайных нейронов другой популяции.
- Условное соединение – определенные нейроны популяций соединяются на основе функции генерации синапса.
- Путем копирования структуры другой проекции.

Пространства имен `parameters_generators`

В пространстве имен `parameters_generators` реализованы функция генерации синапса со значениями атрибутов по умолчанию и класс `CopySynapseGen`. Класс `CopySynapseGen` реализует генератор синапсов, который копирует атрибуты у заданного синапса.

Подробнее о функциях пространства имен `projection` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен `sonata`

В пространстве имен `sonata` реализованы функции для загрузки и сохранения нейронной сети в формате `SONATA`. Пространство имен предоставляет следующие функции:

`load_network()`

Загружает нейронную сеть. Топология нейронной сети должна быть описана в формате `SONATA`.

`save_network()`

Сохраняет топологию и веса нейронной сети в формате `SONATA`.

Подробнее о функциях пространства имен `sonata` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен storage

Пространство имен `storage` содержит пространство имен `native`, в котором реализованы функции для загрузки и сохранения сообщений `SpikeMessage` в [базовом формате KNP](#). Пространство имен предоставляет следующие функции для работы с файлами базового формата:

`load_messages_from_json`

Загружает сообщения из JSON-файла.

`load_messages_from_h5`

Загружает сообщения из HDF5-файла.

`save_messages_to_json`

Сохраняет сообщения в JSON-файл.

`save_messages_to_h5`

Сохраняет сообщения в HDF5-файл.

Подробнее о функциях пространства имен `storage` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Пространство имен synchronization

В пространстве имен `synchronization` реализована функция `get_network_copy()` для получения актуального состояния нейронной сети `Network` с бэкенда `Backend`. Например, вы можете использовать эту функцию при изменении качества предсказаний нейронной сети или через каждые несколько шагов ее исполнения. Вы можете сравнить качество сохраненных копий нейронной сети для выбора наиболее оптимальной для ее загрузки на устройство.

Класс BackendLoader

Класс `BackendLoader` реализует загрузчик экземпляров бэкенда `Backend`. Указатель на экземпляр бэкенда передается в исполнитель модели `ModelExecutor` для ее исполнения на бэкенде.

С помощью методов класса `BackendLoader` вы можете загружать динамические библиотеки и проверять, являются ли загруженные библиотеки бэкендами.

Подробнее о методах класса `BackendLoader` см. в [API-документации Kaspersky Neuromorphic Platform](#).

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

Класс Model

Класс `Model` реализует модель. Модель хранит нейронную сеть `Network`, канал ввода `InputChannel` и канал вывода `OutputChannel` и связывает их друг с другом.

При создании модели необходимо указать ранее созданную нейронную сеть `Network`. После создания модели в нее необходимо добавить канал ввода `InputChannel` и канал вывода `OutputChannel`. Вы можете добавить каналы ввода и вывода путем их подключения к проекциям `Projection` и популяциям `Population`, используемым в нейронной сети.

С помощью методов класса `Model` вы также можете получить нейронную сеть и каналы ввода и вывода, используемые в модели, а также получить базовые данные модели.

Для исполнения модели на разных бэкендах вы можете загрузить модель в исполнитель моделей `ModelExecutor` и вызвать метод `start()`.

Подробнее о методах модели `Model` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Класс ModelExecutor

Класс `ModelExecutor` реализует исполнитель моделей. При создании исполнителя моделей необходимо передать модель `Model`, экземпляр бэкенда `Backend`, который будет исполнять модель, и карту каналов ввода `InputChannel`. Каждый исполнитель моделей содержит следующие атрибуты:

- Загрузчика моделей `ModelLoader`.
- Наблюдателей `MessageObserver`.

С помощью методов класса `ModelExecutor` вы можете исполнить загруженную модель на разных бэкендах. Для исполнения модели класс `ModelExecutor` использует загрузчик моделей `ModelLoader`, а также указатель на экземпляр бэкенда `Backend`, полученный от объекта класса `BackendLoader`. Помимо исполнения модели, с помощью методов класса `ModelExecutor` вы можете выполнить следующие действия:

- Добавить в исполнитель моделей `ModelExecutor` ранее созданного наблюдателя `MessageObserver`. Наблюдателя `MessageObserver` необходимо добавить для получения сообщений по подписке `Subscription`.
- Добавить в исполнитель моделей `ModelExecutor` обработчик сообщений `SpikeMessage` из пространства имен `modifier`.
- Запустить и остановить обучение модели `Model`.
- Получить загрузчик моделей `ModelLoader` и указатель на экземпляр бэкенда `Backend`.

Подробнее о методах класса `ModelExecutor` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Класс ModelLoader

Класс `ModelLoader` реализует загрузчик моделей, который загружает модель `Model` в нужный экземпляр бэкенда `Backend`. При создании загрузчика моделей необходимо указать карту канала ввода `InputChannel` и указатель на экземпляр бэкенда `Backend`. После создания загрузчика моделей вы можете вызвать метод `load()` для загрузки модели на бэкенд. Предварительно модель необходимо создать, передав в конструктор модели нейронную сеть `Network`, хранящую популяции нейронов `Population` и проекции синапсов `Projection`.

Класс `ModelLoader` предоставляет методы, которые вы можете использовать для получения канала ввода `InputChannel` и канала вывода `OutputChannel`, используемых в модели `Model`, а также указателя на загруженный экземпляр бэкенда `Backend`.

Каждый загрузчик моделей содержит следующие атрибуты:

- Экземпляр бэкенда `Backend`, на который требуется загрузить модель.
- Карту канала ввода `InputChannel`, содержащую идентификатор `UID`, [хешированный идентификатор](#) и [преобразователь входных данных](#).
- Канал ввода `InputChannel` и канал вывода `OutputChannel`.

Подробнее о методах загрузчика моделей `ModelLoader` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Класс Network

Класс `Network` реализует нейронную сеть. В нейронной сети хранятся популяции `Population`, проекции `Projection`. Все популяции сети связаны проекциями.

Класс `Network` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- добавлять в нейронную сеть созданные ранее популяции `Population` и проекции `Projection`;
- удалять из нейронной сети ненужные популяции `Population` и проекции `Projection` с заданными уникальными идентификаторами `UID`;
- связывать популяции `Population` и проекции `Projection`, добавленные в нейронную сеть, между собой;
- проверять, содержит ли нейронная сеть популяцию `Population` с заданным уникальным идентификатором;
- проверять, содержит ли нейронная сеть проекцию `Projection` с заданным уникальным идентификатором;
- получать популяции `Population` и проекции `Projection` в составе нейронной сети;

- получать базовые данные `BaseData` нейронной сети;
- получать информацию о количестве популяций `Population` и проекций `Projection` в составе нейронной сети.

После того, как вы добавили в нейронную сеть популяции `Population` и проекции `Projection` и связали их между собой, вы можете передать нейронную сеть в модель `Model` для ее создания.

Подробнее о методах нейронной сети `Network` см. в [API-документации Kaspersky Neuromorphic Platform](#).

Решение типовых задач на C++

Этот раздел содержит описание типовых пользовательских задач, выполняемых на языке программирования C++.

Дополнительно вы можете найти следующие примеры использования платформы в директории `/usr/src/knp/examples` после [установки](#) пакета `knp-examples` или в репозитории платформы:

- `mnist-client` – пример классификации изображений рукописных чисел из базы данных MNIST.
- `mnist-learn` – пример обучения нейронной сети на изображениях рукописных чисел и их меток из базы данных MNIST и ее исполнения на многопоточном бэкенде для CPU. В примере для нейронной сети используется архитектура CoLaNET.
- `save-load-networks` – пример сохранения и загрузки нейронной сети в формате [SONATA](#).
- `simple-network` – пример создания и исполнения нейронной сети, загруженной на бэкенд автоматически.

В директории с каждым примером содержится файл `README.md`, в котором представлено описание примера.

Добавление нового типа нейрона

На текущий момент Kaspersky Neuromorphic Platform поддерживает наборы свойств и типизацию популяций по моделям нейронов `AltAILIF`, `BLIFATNeuron` и `SynapticResourceSTDPBLIFATNeuron`. Если требуется, вы можете добавить в библиотеку [neuron_traits](#) новый тип нейрона.

Вы можете использовать эту инструкцию при работе с исходным кодом платформы.

Чтобы добавить новый тип нейрона:

1. Из рабочей директории с исходным кодом платформы перейдите в директорию `neuron-trait-library/include/knp/neuron-traits/` и создайте в ней 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>
    {
        // Указание количества шагов нейронной сети с шага последнего спайка по умолчанию.
        constexpr static std::size_t n_time_steps_since_last_firing_ =
        std::numeric_limits<std::size_t>::infinity();
        // Указание значения по умолчанию, к которому стремится мембранный потенциал для
        // тормозных синапсов, основанных на проводимости.
        constexpr static double reverse_inhibitory_potential = -0.3;
        // Указание значения по умолчанию, к которому стремится мембранный потенциал для
        // тормозных синапсов, основанных на токе.
        constexpr static double min_potential = -1.0e9;
        // ...
    };

    template <>
    struct neuron_parameters<BLIFATNeuron>
    {
        // Указание количества шагов нейронной сети с шага последнего спайка.
        std::size_t n_time_steps_since_last_firing_ =
        default_values<BLIFATNeuron>::n_time_steps_since_last_firing_;
        // Указание порогового значения мембранного потенциала нейрона.
        double activation_threshold_ = default_values<BLIFATNeuron>::activation_threshold_;
        // Указание динамического порогового значения мембранного потенциала, после достижения
        // которого нейрон генерирует спайк.
        double dynamic_threshold_ = default_values<BLIFATNeuron>::dynamic_threshold_;
        // ...
    };
} // 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 BLIFATNeuron
    // ...
} // namespace knp::neuron_traits
```

4. Реализуйте в нужных бэкендах перегруженный метод запуска.

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

Добавление нового типа синапса

На текущий момент Kaspersky Neuromorphic Platform поддерживает наборы свойств и типизацию проекций по моделям синапсов `DeltaSynapse`, `AdditiveSTDPDeltaSynapse` и `SynapticResourceSTDPDeltaSynapse`. Если требуется, вы можете добавить в библиотеку [synapse_traits](#) новый тип синапса.

Вы можете использовать эту инструкцию при работе с исходным кодом платформы.

Чтобы добавить новый тип синапса:

1. Из рабочей директории с исходным кодом платформы перейдите в директорию `synapse-traits-library/include/knp/synapse-traits/` и создайте заголовочный файл для нового типа синапса.
2. В созданном заголовочном файле определите структуру типа синапса и его шаблонные свойства (например, `default_values`, `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 default_values<DeltaSynapse>
    {
        // Указание значения веса синапса по умолчанию.
        constexpr static float weight_ = 0.0F;
        // Указание значения задержки синапса по умолчанию.
        constexpr static uint32_t delay_ = 1;
        // Указание типа синапса по умолчанию.
        constexpr static OutputType output_type_ = OutputType::EXCITATORY;
    };

    template <>
    struct synapse_parameters<DeltaSynapse>
    {
        // Указание атрибутов 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 DeltaSynapse

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

4. Реализуйте в нужных бэкендах перегруженный метод запуска.

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

Подготовка модели и ее исполнение на бэкенде

Для исполнения модели на бэкенде требуется предварительно создать необходимые объекты платформы и связать их между собой.

Вы можете использовать этот сценарий при разработке прикладных решений.

Сценарий подготовки модели к исполнению состоит из следующих этапов:

1 Создание популяций

Для создания популяций вы можете использовать класс `Population` или функции пространства имен `population`. При создании в конструктор популяции или функцию генерации популяции передается генератор нейронов. Если требуется, вы можете удалить сгенерированные нейроны из популяций или добавить в них новые нейроны.

2 Создание проекций

Для создания проекций вы можете использовать класс `Projection` или функции пространства имен `projection`. При создании в конструктор проекции или функцию генерации проекции передается генератор синапсов. Если требуется, вы можете удалить сгенерированные синапсы из проекций или добавить в них новые синапсы.

3 Создание нейронной сети

Создайте экземпляр класса `Network`.

С помощью методов `add_population()` и `add_projection()` класса `Network` добавьте в нейронную сеть созданные ранее популяции и проекции. При необходимости вы можете дополнительно связать между собой популяции, добавленные в нейронную сеть с помощью метода `connect_populations()`.

4 Создание модели

Создайте экземпляр класса `Model`. При создании модели передайте в конструктор ранее созданную нейронную сеть.

Сценарий исполнения модели на бэкенде состоит из следующих этапов:

1 Загрузка бэкенда

Создайте загрузчик бэкенда с помощью класса `BackendLoader`. Для загрузки бэкенда используйте метод `load()`, передав в него путь к динамической библиотеке бэкенда. Метод возвращает указатель на экземпляр бэкенда `Backend`, который необходимо передать в исполнитель моделей `ModelExecutor`.

После загрузки бэкенда автоматически выполняется его инициализация. При инициализации бэкенда автоматически создаются объекты шины сообщений `MessageBus` и точек подключений `MessageEndpoint` к шине сообщений.

Если требуется, вы можете исполнить несколько разных моделей на загруженном бэкенде. Вы также можете загрузить несколько бэкендов для исполнения на них одной или нескольких моделей.

2 Запуск исполнения модели

Для исполнения модели требуется предварительно создать исполнитель моделей с помощью класса `ModelExecutor`. При создании исполнителя моделей передайте в него ранее созданную модель `Model`, указатель на загруженный экземпляр бэкенда `Backend` и канал ввода `InputChannel`. При необходимости создайте наблюдателя `MessageObserver` и добавьте его в исполнитель моделей `ModelExecutor` для получения сообщений по подписке.

После загрузки модели `Model` в исполнитель моделей `ModelExecutor` автоматически выполняются следующие шаги:

1. Выполняется загрузка модели `Model` на бэкенд `Backend`.
2. Фреймворк создает объекты канала ввода `InputChannel` и канала вывода `OutputChannel` и добавляет их в модель `Model`.
3. Бэкенд `Backend` автоматически подписывает объекты модели друг на друга.

Для запуска исполнения модели используйте метод `start()`. При исполнении модели на бэкенде передача сообщений до точки подключения через шину сообщений осуществляется автоматически.

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 бэкенд для CPU и исполнению нейронной сети.

Этот вариант построения и исполнения нейронной сети используется редко. Вы можете использовать эту инструкцию при разработке прикладных решений.

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 бэкенде для CPU:

C++

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

#include <vector>

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

```
// ...
// Реализация функции, генерирующей нейроны.
// Функция хранится в переменной neuron_generator.
auto neuron_generator = [](size_t index)
{
    return knpp::neuron_traits::neuron_parameters<knpp::neuron_traits::BLIFATNeuron>{};
};
// ...
```

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

Пример реализации генератора дельта-синапсов:

C++

```
// ...
// Реализация функции, генерирующей 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, knpp::synapse_traits::OutputType::EXCITATORY}, 0, 0};
};

// Реализация функции, генерирующей 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, knpp::synapse_traits::OutputType::EXCITATORY}, 0, 0};
};
// ...
```

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

Пример создания объекта популяции:

```
C++

// ...
// Создание объекта популяции.
BLIFATPopulation population{neuron_generator, 1};
```

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

Пример создания проекции, замыкающей вывод популяции на себя:

```
C++

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

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

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

```
C++

// ...
// Создание объекта 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
};

// Получение 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 бэкэнд `Backend`.

Пример загрузки популяции и проекций на бэкэнд:

```
C++

// ...
Backend backend;
// ...
// Загрузка на бэкэнд созданного объекта популяции.
backend.load_populations({population});

// Загрузка на бэкэнд созданных объектов проекций.
backend.load_projections({input_projection, loop_projection});
```

9. Определите объект точки подключения `MessageEndpoint` к шине сообщений.

Пример создания точки подключения к шине сообщений:

```
C++

// ...
// Создание точки подключения к шине сообщений.
auto endpoint = backend.message_bus_.create_endpoint();
// ...
```

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

Пример подключения к каналам ввода и вывода:

```
C++

// ...
// Создание произвольного UID канала ввода.
knp::core::UID in_channel_uid;

// Создание произвольного UID канала вывода.
knp::core::UID out_channel_uid;

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

// Создание подписки 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)
{
    // Отправка спайков 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();
    // Получение спайков, отправленных популяцией на канал вывода.
    auto output = endpoint.unload_messages<knp::core::messaging::SpikeMessage>(out_channel_uid);

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

[Шаги исполнения нейронной сети](#) 

Исполнение нейронной сети в примере 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 в примере.

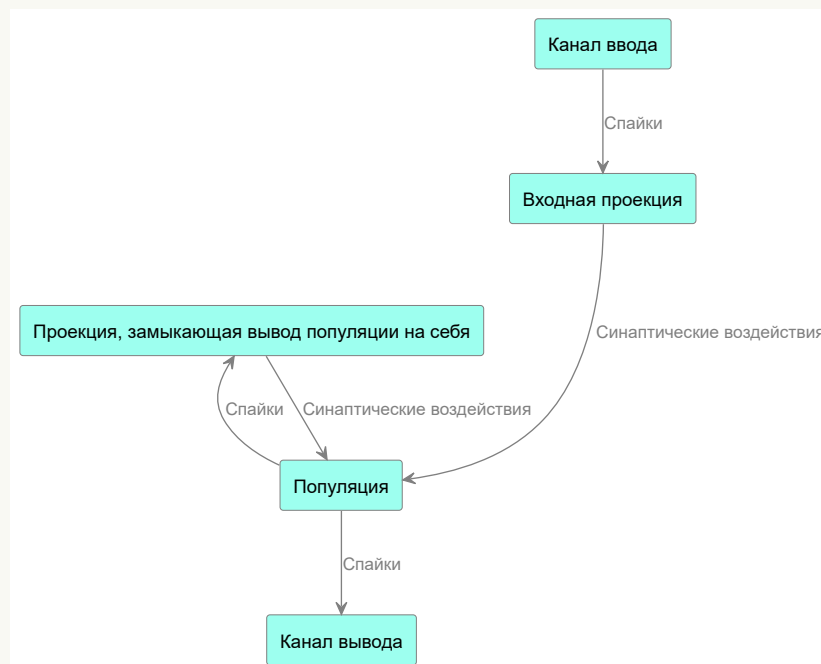


Схема исполнения нейронной сети в примере

В этом примере нейронная сеть выполняется 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.

12. Сохраните созданный файл.

Если требуется, [соберите прикладное решение](#) с помощью CMake.

Компоненты платформы для использования на Python

Этот раздел содержит описание компонентов для использования Kaspersky Neuromorphic Platform на языке программирования Python.

Модуль neuron_traits

Модуль `neuron_traits` представляет собой набор атрибутов [нейронов](#). Вы можете использовать атрибуты нейронов, представленных в модуле, например, при построении популяций.

На текущий момент модуль поддерживает наборы свойств для следующих нейронов:

- Нейрон по модели [AltAILIF](#).
- Нейрон по модели [BLIFATNeuron](#).
- Нейрон по модели [SynapticResourceSTDPBLIFATNeuron](#).

Если требуется, при [работе с исходным кодом](#) вы можете реализовать и использовать собственные модели нейронов.

Модуль synapse_traits

Модуль `synapse_traits` представляет собой набор атрибутов [синапсов](#). Вы можете использовать синапсы, представленные в модуле, например, при построении проекций.

На текущий момент модуль поддерживает наборы свойств для следующих синапсов:

- Синапс по модели [DeltaSynapse](#).
- Синапс по модели [SynapticResourceSTDPDeltaSynapse](#).

Если требуется, при [работе с исходным кодом](#) вы можете реализовать и использовать собственные модели синапсов.

Модуль core

Модуль `core` предоставляет классы, необходимые для взаимодействия с бэкендом и выполнения прикладных задач.

Таблица ниже содержит описание компонентов модуля.

Компонент	Описание
messaging	Модуль, содержащий интерфейсы к сообщениям.
Backend	Базовый класс для конкретных реализаций бэкендов.
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 к шине сообщений.
Population	Набор классов, реализующих контейнеры нейронов одной модели.
Projection	Набор классов, реализующих контейнеры синапсов одного типа.
Subscription	Набор классов, реализующих подписку на сообщения.
TagMap	Класс, реализующий интерфейс к словарию тегов.
UID	Класс, определяющий уникальные идентификаторы.
uid_hash	Класс, реализующий хеширования UID.
UUID	Класс, реализующий интерфейс к уникальному идентификатору библиотеки Boost:Uuid .

На рисунке ниже представлена схема взаимодействия компонентов модуля.

В Kaspersky Neuromorphic Platform структура [BaseData](#) содержит базовую информацию об объекте и может включать в себя [UID](#) и [TagMap](#).

Объекты классов [Population](#) и [Projection](#) хранят данные, которые бэкенд обрабатывает и передает другим объектам платформы посредством сообщений, реализованных в классах модуля [messaging](#). Сообщения содержат в своем заголовке UID отправителя. Обмен сообщениями осуществляется с помощью методов класса [Backend](#), в котором хранятся классы [MessageBus](#) и [MessageEndpoint](#), а также устройство [Device](#). Для подключения к шине сообщений [MessageBus](#) объекты платформы создают точки подключения [MessageEndpoint](#), с которыми шина сообщений взаимодействует с ними в процессе обмена сообщениями. Каждая точка подключения [MessageEndpoint](#) содержит контейнер подписок класса [Subscription](#).

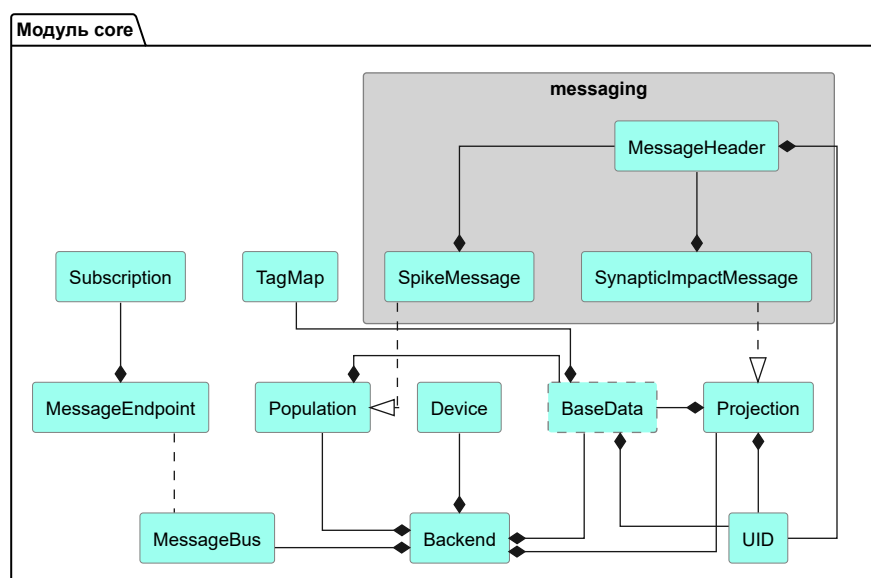


Схема взаимодействия компонентов модуля core

Модуль messaging

Модуль `messaging` реализует интерфейсы для доступа к сообщениям. Каждое сообщение содержит следующий набор свойств:

- `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 свойств зависит от типа сообщения.

Обмен сообщениями обеспечивается с помощью шины сообщений `MessageBus`. С помощью сообщений происходит обновление пользовательских структур и синхронизация изменений на бэкенде. Таблица ниже содержит описание поддерживаемых типов сообщений бэкенда.

Структуры сообщений

Тип сообщения	Описание	Структура сообщения
<code>SpikeMessage</code>	Сообщение к проекции синапсов, содержащее спайк [?] . На сообщения, содержащие спайки, может быть подписано более одной проекции.	<code>header</code> – заголовок сообщения, содержащий UID канала ввода или популяции, нейроны которой сгенерировали спайки, и время генерации спайков. <code>neuron_indexes</code> – уникальные индексы нейронов популяции, сгенерировавших спайки.
<code>SynapticImpactMessage</code>	Сообщение от проекции синапсов к популяции нейронов, содержащее синаптическое воздействие [?]	<code>header</code> – заголовок сообщения, содержащий UID проекции и время отправки сообщения. <code>postsynaptic_population_uid</code> – UID постсинаптической популяции. <code>presynaptic_population_uid</code> – UID пресинаптической популяции. <code>is_forcing</code> – флаг для включения и выключения пластичности в проекции синапсов. Если в проекции синапсов используется синаптическая пластичность, то атрибут имеет значение <code>true</code> , иначе имеет значение <code>false</code> . <code>impacts</code> – массив, содержащий следующие атрибуты: • <code>connection_index</code> – индекс соединения. • <code>impact_value</code> – значение синаптического воздействия. • <code>synapse_type</code> – тип синапса. • <code>presynaptic_neuron_index</code> – уникальный индекс пресинаптического нейрона, от которого был получен спайк синапсом проекции. • <code>postsynaptic_neuron_index</code> – уникальный индекс постсинаптического нейрона, которому будет передано значение синаптического воздействия для изменения значений его атрибутов. Массив <code>impacts</code> может содержать более одного набора перечисленных атрибутов для передачи синаптических воздействий нескольким нейронам одной популяции.

Kaspersky Neuromorphic Platform предоставляет методы для взаимодействия с сообщениями. Например, с помощью метода отправки сообщений класса `MessageEndpoint` вы можете отправить определенное сообщение через точку подключения к шине сообщений.

Класс Backend

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

- управлять обучением и исполнением [импульсной нейронной сети](#) на различных устройствах;
- получать сообщения, содержащие спайки и/или синаптические воздействия, для заданной точки подключения к шине сообщений;
- получать внутренние данные для мониторинга работы нейронной сети;
- выгружать внутренние данные мониторинга работы нейронной сети, полученные от вычислителя.

Если предварительно была создана [модель](#), в которую была передана [нейронная сеть](#), хранящая [популяции](#) и [проекции](#), вы можете использовать метод загрузки модели класса [ModelLoader](#) для ее загрузки на бэкенд перед его инициализацией. Вы также можете загрузить проекции, содержащие [синапсы](#), и популяции, содержащие [нейроны](#), вручную. Для этого вы можете использовать методы загрузки проекций и популяций базового класса Backend или соответствующие методы конкретного класса бэкенда.

В процессе инициализации бэкенда загруженные проекции синапсов подписываются на [спайки](#) от связанных [популяций](#) или [каналов ввода](#), а загруженные популяции подписываются на [синаптические воздействия](#) от связанных [проекций](#). При необходимости вы можете подписать необходимые объекты платформы на [сообщения](#) с помощью метода подписки.

Базовый класс Backend предоставляет методы, с помощью которых вы можете управлять обучением и исполнением нейронной сети.

Каждый объект класса Backend имеет следующие свойства:

- [Уникальный идентификатор](#) бэкенда.
- [Шину сообщений](#). С помощью шины сообщений бэкенд обеспечивает обмен сообщениями между объектами модуля core, а также обмен сообщениями от [модификаторов](#) к [нейронам](#) и [синапсам](#).
- [Точку подключения](#) к шине сообщений.
- Статус исполнения нейронной сети на бэкенде.

Структура BaseData

Структура BaseData определяет набор базовых данных, общих для некоторых объектов платформы. Например, базовые данные BaseData используются в объектах классов [Backend](#), [Population](#) и [Projection](#).

Базовые данные BaseData имеют следующие свойства:

- [Уникальный идентификатор](#) объекта.
- Ассоциативный массив [TagMap](#), состоящий из тегов, используемых объектом, и их значений.

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 `Device`

Класс `Device` является базовым для классов 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.

Класс `MessageBus`

Класс `MessageBus` реализует интерфейс для доступа к шине сообщений. Шина сообщений обеспечивает прием, распределение и доставку сообщений между подключенными к ней объектами.

Класс `MessageBus` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- создавать точки подключения к шине сообщений;
- принимать, распределять и доставлять сообщения между объектами, подключенными к шине сообщений.

Шина сообщений предоставляет множество [точек подключения](#), с помощью которых объекты платформы и объекты [внешней среды](#) могут подписаться на [сообщения](#) или отправить их. Вы можете создать точки подключения к шине сообщений и затем подписать их на сообщения определенного типа с помощью метода подписки экземпляра класса `MessageEndpoint`. 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](#). Вы можете создать точку подключения с помощью соответствующего метода класса [MessageBus](#). Шина сообщений может иметь несколько точек подключения.

Точка подключения хранит в себе идентификатор получателя сообщений и ключ класса подписки, который используется для фильтрации [сообщений](#). Точка подключения может хранить в себе набор подписок на сообщения от различных отправителей и для различных получателей.

Класс `MessageEndpoint` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- Подписывать точки подключения на сообщения определенного типа.

- Удалять подписки на сообщения от определенных отправителей.

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

- Получать сообщения от отправителя, на который подписана точка подключения.

Перед вызовом метода необходимо использовать метод маршрутизации сообщений объекта класса `MessageBus` для получения сообщений точкой подключения.

- Отправлять сообщения с помощью шины сообщений.

Набор классов Population

Набор классов `Population` реализует контейнеры, каждый из которых содержит [нейроны](#), принадлежащие одной модели. Для инициализации популяции необходимо указать количество нейронов и генератор нейронов определенного типа.

Вы можете загрузить популяции на бэкенд для вычисления нейронной сети. Если модель, хранящая популяции, загружена на бэкенд, то загрузка популяций происходит автоматически с помощью загрузчика моделей `ModelLoader`. Вы также можете вручную передать данные популяции на бэкенд с помощью метода загрузки популяций базового класса `Backend`.

После вычисления новые свойства популяций хранятся на бэкенде. Для получения от бэкенда текущих параметров нейронной сети вы можете вызвать синхронизацию. Если вы вручную изменили свойства популяции до вызова синхронизации, вы можете заново загрузить популяцию на бэкенд для вычисления нейронной сети.

Набор классов `Population` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- добавлять нейроны в ранее созданную популяцию;
- удалять один или несколько нейронов из популяции;
- получать уникальный идентификатор популяции;
- получать атрибуты нейронов популяций;
- получать информацию о количестве нейронов в популяции;
- выполнять итерацию по популяции;
- получать определенный элемент популяции.

Набор классов Projection

Набор классов `Projection` реализует контейнеры, каждый из которых содержит [синапсы](#) одного из типов, поддерживаемых модулем [synapse traits](#). Каждая проекция синапсов имеет уникальный идентификатор.

В классах проекций хранятся синапсы, представляющие собой соединения между нейронами пресинаптической и постсинаптической популяций. Идентификаторы этих популяций необходимо указать при инициализации проекции. С помощью набора классов `Projection` вы можете управлять синапсами в составе проекции, а также передавать соединения между нейронами на бэкэнд и получать их от бэкэнда обратно.

Набор классов `Projection` предоставляет методы, с помощью которых вы можете получить уникальный идентификатор проекции и атрибуты синапсов проекций. Вы также можете получить информацию о количестве синапсов в проекции, выполнить итерацию по проекции и получить определенный элемент проекции.

Набор классов Subscription

Набор классов `Subscription` реализует функциональность подписки на сообщения и хранит в себе поступившие [сообщения](#). Python-фреймворк предоставляет следующие классы подписок для хранения сообщений определенного типа до их запроса получателем:

- `SpikeMessageSubscription` – для подписки на сообщения `SpikeMessage`.
- `SynapticImpactMessageSubscription` – для подписки на сообщения `SynapticImpactMessage`.

Вы можете создать подписку с помощью метода подписки класса [MessageEndpoint](#). Созданные подписки хранятся в точках подключения. Если вы удалите точку подключения, с которой связаны объекты подписки, то эти подписки также будут удалены. Передача сообщений по этим подпискам прекратится.

Подписка обеспечивает [фильтрацию сообщений](#). В объекте подписки определены идентификаторы получателя и отправителей сообщений, в соответствии с которыми шина сообщений распределяет сообщения и доставляет их до получателей.

Набор классов `Subscription` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- добавлять отправителей в подписку;
- удалять отправителей из подписки;
- проверять, связана ли подписка с определенным отправителем.

Класс TagMap

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

Класс UID

Класс `UID` реализует уникальный идентификатор объекта. Идентификатор объекта часто используется как часть структуры [BaseData](#).

Если требуется, вы можете сгенерировать случайный UID или создать уникальный идентификатор [UUID](#), а также получить хеш-значение идентификатора объекта. Класс `UID` также предоставляет логические операторы и операторы сравнения и присваивания.

Класс uid_hash

Класс `uid_hash` реализует метод хеширования идентификаторов объектов.

Класс UUID

Класс `UUID` реализует интерфейс к уникальному идентификатору библиотеки [Boost:Uuid](#). Каждый уникальный идентификатор имеет следующие свойства:

- Размер уникального идентификатора в байтах.
- Функцию, возвращающую значение `true`, если уникальный идентификатор состоит исключительно из нулей.
- Один из следующих вариантов идентификатора:
 - `NCS` – вариант, зарезервированный для обратной совместимости с форматом UUID Apollo Network Computing System 1.5.
 - `RFC_4122` – вариант, описанный в RFC 4122.
 - `MICROSOFT` – вариант, зарезервированный для обратной совместимости с ранними GUID Microsoft Windows.
 - `FUTURE` – вариант, зарезервированный для использования в будущем.

Python-фреймворк

Python-фреймворк предназначен для создания [импульсных нейронных сетей](#) и также преобразования [искусственных нейронных сетей](#) в импульсные. Созданные импульсные нейронные сети обучаются методами синаптической пластичности и могут быть исполнены на центральном процессоре, [нейроморфном процессоре Алтай-2](#) или его программном эмуляторе. Искусственные нейронные сети, преобразованные в импульсные с помощью тернарных слоев пакета [ANN2SNN](#), обучаются методом обратного распространения ошибки с учетом ограничений нейроморфного процессора Алтай-2. Полученные в результате нейронные сети адаптированы для исполнения на нейроморфном процессоре Алтай-2 или его программном эмуляторе.

Python-фреймворк представляет собой блоки, которые обеспечивают управление бэкендами, преобразуют представление нейронной сети в разные форматы и выбирают оптимальную нейронную сеть для запуска.

С помощью Python-фреймворка Kaspersky Neuromorphic Platform вы можете динамически загружать различные бэкенды.

Таблица ниже содержит описание компонентов Python-фреймворка.

Компоненты Python-фреймворка

Компонент	Описание
ANN2SNN	Пакет, реализующий тернарные слои нейронной сети.
knp	Пакет, реализующий Python-фреймворк для работа на бэкенде AltAI SNN .
input	Модуль, в котором реализован интерфейс для доступа к каналу ввода.
output	Модуль, в котором реализован интерфейс для доступа к каналу вывода.
sonata	Модуль, который содержит функции загрузки и сохранения нейронной сети в формате SONATA .
BackendLoader	Класс, реализующий загрузчик динамических библиотек.
ImpactMessageObserver	Класс, реализующий наблюдателя за сообщениями SynapticImpactMessage .
Model	Класс, реализующий модель.
ModelExecutor	Класс, реализующий исполнитель моделей.
ModelLoader	Класс, реализующий загрузчик модели.
Network	Класс, реализующий объект нейронной сети.
SpikeMessageObserver	Класс, реализующий наблюдателя за сообщениями SpikeMessage .

На рисунке ниже представлена схема взаимодействия компонентов Python-фреймворка.

В Kaspersky Neuromorphic Platform объект класса [Network](#) может быть загружен с помощью функции пространства имен [sonata](#). Нейронная сеть, а также контейнеры каналов ввода и вывода, реализованных в модулях [input](#) и [output](#) соответственно, хранятся в объекте класса [Model](#). Каналы ввода и вывода используются для приема сообщений, содержащих спайки, от входного потока данных и популяций нейронной сети.

Вы можете исполнить модель с помощью метода `start()` объекта класса `ModelExecutor`. Исполнитель моделей также содержит наблюдателей `SpikeMessageObserver` и `ImpactMessageObserver`, которые обрабатывают полученные сообщения в соответствии с заданной функцией, и объект класса `ModelLoader`, с помощью методов которого вы можете загружать модель на конкретный экземпляр бэкенда. Для исполнения модели на экземпляре бэкенда исполнителю моделей передается объект бэкенда, полученный от объекта класса `BackendLoader`.

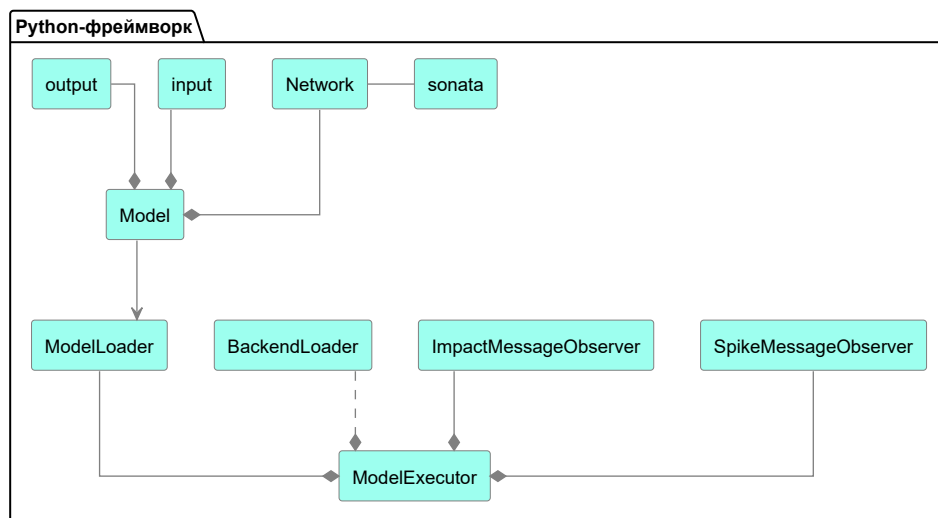


Схема взаимодействия компонентов Python-фреймворка

Пакет ANN2SNN

Пакет `ANN2SNN` предназначен для создания нейронных сетей, которые можно преобразовать в [импульсные нейронные сети](#) и разместить для их исполнения на нейроморфном процессоре Алтай-2. Пакет `ANN2SNN` реализован на языке программирования Python.

Для создания нейронных сетей необходимо использовать специальные [тернарные слои](#), в которых учитываются [ограничения](#), характерные для нейроморфного процессора Алтай-2. Тернарные слои, используемые для создания нейронной сети, соответствуют слоям [библиотеки Keras](#), со следующими основными отличиями:

- Входными и выходными данными каждого из слоев нейронной сети являются бинарные тензоры соответствующей размерности.
- Все веса слоев нейронной сети находятся в диапазоне `{-1, 0, 1}`.
- Значение смещения в слое нейронной сети находится в диапазоне от `-16 384` до `16 383`.

При обучении нейронной сети, созданной с помощью тернарных слоев, стоит учитывать следующие особенности:

- В случае наличия небинаризованных данных для обучения и/или валидации необходимо сначала их привести к бинарному виду. Для это можно использовать алгоритмы кодирования и декодирования или [создать нейросетевые обучаемые блоки, кодирующие и декодирующие данные](#).
- При обучении моделей бинарной классификации методом градиентного спуска для дифференцируемости функции потерь необходимо указать [сигмоидную функцию](#) (англ. sigmoid function) в качестве функции активации выходного слоя. После проведения обучения нужно заменить сигмоидную активацию выходного слоя на активацию с [функцией Хевисайда](#) (англ. Heaviside function).
- При обучении моделей многоклассовой классификации методом градиентного спуска для дифференцируемости функции потерь необходимо указать [многомерную логистическую функцию](#) (англ. softmax function) в качестве функции активации выходного слоя. При размещении такой нейронной сети выходной вектор будет автоматически преобразован к бинарному виду, реализующему индикатор максимальной компоненты значения выходного вектора.

Сериализация и десериализация нейронных сетей проводится стандартными методами, предоставляемыми библиотекой Keras, с указанием параметра `custom_objects` в виде словаря тернарных слоев и функции активации `heaviside`.

Во время размещения нейронная сеть проверяется на соответствие ограничениям процессора. Проверка нейронной сети производится относительно параметров конфигурации нейроморфного процессора, указанных в [YAML-файле](#). В результате размещения выдается информация о количестве ядер, необходимых для запуска нейронной сети на нейроморфном процессоре Алтай-2. Нейронная сеть переводится в формат, поддерживаемый процессором для ее дальнейшего запуска на аппаратном процессоре или его программном эмуляторе. Если нейронную сеть невозможно исполнить на нейроморфном процессоре из-за превышения [допустимого количества ядер](#), выдается соответствующая ошибка.

В нейроморфном процессоре Алтай-2 предусмотрена возможность масштабирования за счет увеличения количества СБИС. По вопросам масштабирования вам нужно обратиться к специалистам "Лаборатории Касперского".

Вы можете провести примерную оценку количества ядер, необходимых для исполнения нейронной сети, по формуле, представленной на рисунке ниже, где:

- x соответствует необходимому количеству ядер. После расчета количества ядер по формуле округлите результат в большую сторону.
- i соответствует порядковому номеру слоя нейронной сети.
- n соответствует количеству слоев нейронной сети.
- w соответствует первой размерности тензора слоя нейронной сети.
- h соответствует второй размерности тензора слоя нейронной сети.
- c соответствует третьей размерности тензора слоя нейронной сети.

$$x = \sum_{i=1}^n \frac{w_i * h_i * c_i}{512}$$

Формула оценки количества ядер

Количество ядер, необходимых для размещения полносвязного слоя `TernaryDense`, зависит от количества входных нейронов в слое. Если количество входных нейронов не больше 512, то вы можете рассчитать необходимое количество ядер по формуле, представленной ниже, где:

- x соответствует необходимому количеству ядер. После расчета количества ядер по формуле округлите результат в большую сторону.
- i соответствует порядковому номеру слоя нейронной сети.
- n соответствует количеству полносвязных слоев нейронной сети.
- U соответствует количеству целевых нейронов.

$$x = \sum_{i=1}^n \frac{U}{512}$$

Формула оценки количества ядер для слоя `TernaryDense`

Если количество входных нейронов больше 512, то вы можете рассчитать необходимое количество ядер по формуле, представленной ниже, где:

- x соответствует необходимому количеству ядер. После расчета количества ядер по формуле округлите результат в большую сторону.
- i соответствует порядковому номеру слоя нейронной сети.
- n соответствует количеству полносвязных слоев нейронной сети.
- U соответствует количеству целевых нейронов.
- N соответствует количеству входных нейронов для слоя.

$$x = \sum_{i=1}^n \frac{U}{512} * (\frac{N}{512} + 1)$$

Формула оценки количества ядер для слоя `TernaryDense`

Для нейронных сетей, созданных на базе пакета `ANN2SNN`, возможны следующие варианты инференса:

- Инференс за количество [тиков](#), равное количеству слоев в нейронной сети. При этом варианте инференса за тик исполнения нейронной сети вычисляется только один слой нейронной сети, а обработанные данные передаются в последующий слой. Время инференса прямо пропорционально количеству слоев нейронной сети.
- Инференс за один тик. При этом варианте инференса каждый слой нейронной сети получает данные от предыдущего слоя и обрабатывает их. Время инференса составляет около 1,5 мс.

Тернарные слои пакета ANN2SNN

В пакете `ANN2SNN` поддерживаются следующие тернарные слои:

Класс	Описание
<code>TernaryDense</code>	Реализует полносвязный слой, аналогом которого является слой <code>Dense</code> библиотеки <code>Keras</code> . При работе с полносвязным слоем рекомендуется указать количество нейронов слоя (параметр <code>units</code>) равным или меньше 512. Если у нейрона слоя более 512 связей, то такой нейрон будет представлен в виде нескольких нейронов на нейроморфном процессоре Алтай-2.
<code>TernaryConv1D</code>	Реализует одномерный сверточный слой, аналогом которого является слой <code>Conv1D</code> библиотеки <code>Keras</code> . При работе с этим слоем рекомендуется установить следующие ограничения: - Размер ядра (параметр <code>kernel_size</code>) – не более 20. - Количество фильтров (параметр <code>filters</code>) – не более 64.
<code>TernaryConv2D</code>	Реализует двумерный сверточный слой, аналогом которого является слой <code>Conv2D</code> библиотеки <code>Keras</code> . При работе с этим слоем рекомендуется следовать следующим ограничениям: - Не использовать ядра большого размера (параметр <code>kernel_size</code>). Оптимальными считаются ядра с размерами [2, 2], [3, 3] и [4, 4]. - Ограничить размер входных изображений до 128x128 и число каналов до 10. - Ограничить количество фильтров (параметр <code>filters</code>) до 32.
<code>TernaryDepthwiseConv1D</code>	Реализует одномерный сверточный слой, аналогом которого является слой <code>DepthwiseConv1D</code> библиотеки <code>Keras</code> . При работе с этим слоем рекомендуется установить следующие ограничения: - Размер ядра (параметр <code>kernel_size</code>) – не более 20. - Количество фильтров (параметр <code>filters</code>) – не более 64.
<code>TernaryDepthwiseConv2D</code>	Реализует двумерный сверточный слой, аналогом которого является слой <code>DepthwiseConv2D</code> библиотеки <code>Keras</code> . При работе с этим слоем рекомендуется следовать следующим ограничениям: - Не использовать ядра большого размера (параметр <code>kernel_size</code>). Оптимальными считаются ядра с размерами [2, 2], [3, 3] и [4, 4]. - Ограничить размер входных изображений до 128x128 и число каналов до 10. - Ограничить количество фильтров (параметр <code>filters</code>) до 32.
<code>MaxPool1D</code>	Поддерживается стандартная реализация слоя <code>MaxPool1D</code> библиотеки <code>Keras</code> . Для уменьшения количества оборудования нейроморфного процессора, необходимого для исполнения нейронной сети, рекомендуется отказаться от использования этого слоя путем увеличения параметра <code>stride</code> в сверточных слоях.
<code>MaxPool2D</code>	Поддерживается стандартная реализация слоя <code>MaxPool2D</code> библиотеки <code>Keras</code> . Для уменьшения количества оборудования нейроморфного процессора, необходимого для исполнения нейронной сети, рекомендуется отказаться от использования этого слоя путем увеличения параметра <code>stride</code> в сверточных слоях.
<code>BatchNormalization</code>	Поддерживается стандартная реализация нормирующего слоя <code>BatchNormalization</code> библиотеки <code>Keras</code> . Параметр масштабирования <code>scale</code> не поддерживается и должен быть определен как <code>false</code> . Слой <code>BatchNormalization</code> должен идти после слоя <code>Activation</code> .
<code>InputLayer</code>	Поддерживается стандартная реализация входного слоя <code>InputLayer</code> библиотеки <code>Keras</code> . Входные данные должны иметь бинарный тип.
<code>Flatten</code>	Поддерживается стандартная реализация слоя <code>Flatten</code> библиотеки <code>Keras</code> .
<code>Reshape</code>	Поддерживается стандартная реализация слоя <code>Reshape</code> библиотеки <code>Keras</code> .
<code>Concatenate</code>	Поддерживается стандартная реализация слоя <code>Concatenate</code> библиотеки <code>Keras</code> .

Для размещения нейронной сети на нейроморфном процессоре Алтай-2 в качестве функции активации каждого слоя необходимо указать функцию `heaviside`. При обучении нейронной сети методом градиентного спуска используется [кусочно-линейная аппроксимация сигмоидной функции](#) (англ. `hard sigmoid function`).

Модуль input

В модуле `input` реализован интерфейс для доступа к каналу ввода `InputChannel`, который принимает сообщения `SpikeMessage`. При инициализации канала ввода необходимо указать уникальный идентификатор `UID` канала ввода, точку подключения к шине сообщений, с помощью которой данные отправляются подписчикам, а также функтор для генерации сообщений `SpikeMessage`. При наличии подписки сгенерированные сообщения могут быть переданы подписчикам через точки подключения.

Если вы загрузили модель, хранящую нейронную сеть, каналы ввода и вывода, в исполнитель моделей, то при ее исполнении на бэкенде автоматически осуществляется передача отправленных сообщений.

Модуль output

В модуле `output` реализован интерфейс для доступа к каналу вывода `OutputChannel`. Канал вывода принимает набор сообщений, содержащих спайки, от связанной популяции за несколько тактов исполнения нейронной сети.

При инициализации канала вывода необходимо указать уникальный идентификатор `UID` канала вывода и точку подключения к шине сообщений.

Сообщения, полученные каналом вывода, необходимо преобразовать в нужный вам формат данных.

Модуль sonata

В модуле `sonata` реализованы функции для загрузки и сохранения нейронной сети в формате `SONATA`. Пространство имен предоставляет следующие функции:

`load_network()`

Загружает нейронную сеть.

`save_network()`

Сохраняет топологию и веса нейронной сети.

Класс BackendLoader

Класс `BackendLoader` реализует загрузчик экземпляров бэкенда. Экземпляр бэкенда передается в исполнитель модели `ModelExecutor`.

Класс `BackendLoader` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- загружать бэкенды;
- проверять, являются ли загруженные библиотеки бэкендами.

Класс ImpactMessageObserver

Класс `ImpactMessageObserver` реализует наблюдателя, который подписывается на сообщения `SynapticImpactMessage`. При инициализации наблюдателя необходимо указать точку подключения к шине сообщений, которая обеспечивает подписку на сообщения, функцию для обработки полученных сообщений и идентификатор наблюдателя.

С помощью метода подписки вы можете подписать наблюдателя на сообщения `SynapticImpactMessage`. В конце каждого такта исполнения нейронной сети наблюдатель получает сообщения, пришедшие по подписке, и обрабатывает их в соответствии с заданной функцией.

Класс Model

Класс `Model` реализует модель. Модель хранит связи каналов ввода и вывода с популяциями и проекциями. При инициализации модели необходимо указать ранее созданную нейронную сеть. После создания модели в нее необходимо добавить каналы ввода и вывода.

Класс `Model` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- добавлять в модель каналы ввода и вывода;
- получать нейронную сеть, используемую в модели;
- получать каналы ввода и вывода, используемые в модели;
- получать уникальный идентификатор модели.

Для исполнения модели на разных бэкендах вы можете загрузить модель в объект класса `ModelExecutor` и вызвать метод исполнения модели.

Класс ModelExecutor

Класс `ModelExecutor` реализует исполнитель моделей. При инициализации исполнителя моделей необходимо указать модель, которую необходимо исполнить, экземпляр бэкенда, который будет исполнять модель, и карту каналов ввода.

Класс `ModelExecutor` предоставляет методы, с помощью которых вы можете исполнять загруженную модель. Для исполнения модели класс `ModelExecutor` использует экземпляр бэкенда, полученный от объекта класса `BackendLoader`. Помимо исполнения модели, с помощью методов класса `ModelExecutor` вы также можете выполнять следующие действия:

- добавлять наблюдателя за сообщениями определенного типа в исполнитель модели;
- запускать и останавливать обучение модели;

- получать экземпляр бэкенда, используемого для исполнения модели;
- получать канал вывода.

Класс ModelLoader

Класс `ModelLoader` реализует загрузчик моделей, который загружает [модель](#) в нужный экземпляр [бэкенда](#).

Класс `ModelLoader` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- Создавать загрузчики моделей.
- Загружать модель на бэкенд.

Предварительно модель необходимо создать, передав в конструктор модели [нейронную сеть](#), хранящую [популяции](#) нейронов и [проекции](#) синапсов.

- Получать [каналы ввода](#) и [вывода](#), используемые в модели.

Класс Network

Класс `Network` реализует нейронную сеть. Нейронная сеть хранит [популяции](#), [проекции](#), а также обеспечивает общий интерфейс для доступа к нейронам и синапсам нейронной сети. В нейронной сети все популяции связаны проекциями.

Каждая нейронная сеть имеет следующие свойства:

- Количество популяций в нейронной сети.
- Количество проекций в нейронной сети.

Класс `Network` предоставляет методы, с помощью которых вы можете выполнять следующие действия:

- добавлять популяции и проекции в нейронную сеть;
- удалять популяции и проекции из нейронной сети;
- получать уникальный идентификатор нейронной сети;
- получать популяции и проекции в составе нейронной сети;
- получать диапазоны итераторов популяций и проекций в нейронной сети;
- получать информацию о количестве популяций и проекций в нейронной сети.

Класс SpikeMessageObserver

Класс `SpikeMessageObserver` реализует наблюдателя, который подписывается на сообщения `SpikeMessage`. При инициализации наблюдателя необходимо указать точку подключения к шине сообщений, которая обеспечивает подписку на сообщения, функцию для обработки полученных сообщений и идентификатор наблюдателя.

С помощью метода подписки вы можете подписать наблюдателя на сообщения `SpikeMessage`. Во время исполнения нейронной сети наблюдатель получает сообщения, пришедшие по подписке, и обрабатывает их в соответствии с заданной функцией.

Решение типовых задач на Python

Этот раздел содержит описание типовых пользовательских задач, выполняемых на языке программирования Python.

Для Python предоставлен пример `mnist-client`, содержащий пример классификации изображений рукописных чисел из базы данных MNIST.

Исполнение нейронной сети, загруженной на бэкенд вручную

Этот раздел содержит инструкции по созданию нейронной сети, ее загрузке на бэкенд для CPU и исполнению.

Вы можете использовать эту инструкцию для разработки прикладных решений.

Чтобы автоматически загрузить нейронную сеть на бэкенд и исполнить ее:

1. В директории вашего проекта создайте файл программы с расширением `.py`, в котором будет реализована функция для создания и запуска нейронной сети.
2. В файле программы импортируйте объекты библиотек платформы, необходимые для исполнения нейронной сети.

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

Пример подключения объектов библиотек для исполнения нейронной сети с популяцией BLIFAT-нейронов и проекцией дельта-синапсов на однопоточном бэкенде для CPU:

Python

```
from knp.base_framework._knp_python_framework_base_framework import BackendLoader
from knp.core._knp_python_framework_core import UID, BLIFATNeuronPopulation, DeltaSynapseProjection,
SpikeMessage
from knp.neuron_traits._knp_python_framework_neuron_traits import BLIFATNeuronParameters
from knp.synapse_traits._knp_python_framework_synapse_traits import DeltaSynapseParameters, OutputType
```

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

Пример реализации генератора BLIFAT-нейронов:

Python

```
# Реализация функции, генерирующей BLIFAT-нейроны.
def neuron_generator(): # type: ignore[no-untyped-def]
    return BLIFATNeuronParameters()
```

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

Пример реализации генератора дельта-синапсов:

Python

```
# Реализация функции, генерирующей дельта-синапсы для проекции, которая будет замыкать вывод
# популяции на себя.
def synapse_generator(_): # type: ignore[no-untyped-def]
    return DeltaSynapseParameters(1.0, 6, OutputType.EXCITATORY), 0, 0

# Реализация функции, генерирующей дельта-синапсы для проекции, которая будет связана
# с каналом ввода.
def input_projection_gen(_): # type: ignore[no-untyped-def]
    return DeltaSynapseParameters(1.0, 1, OutputType.EXCITATORY), 0, 0
```

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

Пример создания функции main:

Python

```
def main(): # type: ignore[no-untyped-def]
    # ...
```

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

Пример создания объекта популяции:

Python

```
# ...
# Создание объекта популяции с одним BLIFAT-нейроном.
population = BLIFATNeuronPopulation(neuron_generator, 1)
# ...
```

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:

Python

```
# ...
# Создание объекта проекции с одним дельта-синапсом, замыкающей вывод популяции
# на себя.
loop_projection = DeltaSynapseProjection(
    population.uid, population.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:

Python

```
# ...
# Создание объекта входной проекции с одним дельта-синапсом, которой присваивается
# нулевой идентификатор (UID(False)). Проекция принимает спайки от канала ввода
# и отправляет синаптическое воздействие в объект популяции.
input_projection = DeltaSynapseProjection(
    UID(False), population.uid, input_projection_gen, 1
)

# Сохранение UID входной проекции в переменной input_uid.
input_uid = input_projection.uid
# ...
```

9. В созданной функции загрузите экземпляр бэкенда и загрузите в него созданные популяцию и проекции. Создайте точку подключения.

Пример загрузки бэкенда и создания точки подключения:

Python

```
# ...
# Загрузка экземпляра однопоточного бэкенда для CPU с локального компьютера.
backend = BackendLoader().load(f'{pytestconfig.rootdir}/../lib/libknp-cpu-single-threaded-backend')

# Загрузка созданной популяции в экземпляр бэкенда.
backend.load_all_populations([population])
# Загрузка созданных проекций в экземпляр бэкенда.
backend.load_all_projections([input_projection, loop_projection])

# Инициализация бэкенда.
backend._init()
# Создание объекта точки подключения.
endpoint = backend.message_bus.create_endpoint()
# ...
```

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

Пример определения идентификаторов каналов ввода и вывода:

Python

```
# ...
# Создание произвольных идентификаторов in_channel_uid и out_channel_uid для каналов
# ввода и вывода.
in_channel_uid = UID()
out_channel_uid = UID()
# ...
```

11. В созданной функции подпишите бэкенд и точку подключения на сообщения от каналов ввода и вывода.

Пример подписки бэкенда и точки подключения на сообщения `SpikeMessage`:

Python

```
# ...
# Подписка бэкенда на спайки от канала ввода.
backend.subscribe(SpikeMessage, input_uid, [in_channel_uid])
# Подписка точки подключения на спайки от канала вывода.
endpoint.subscribe(SpikeMessage, out_channel_uid, [population.uid])
# ...
```

12. В созданной функции исполните нейронную сеть. При необходимости создайте список, в который будут записываться индексы шагов, на которых на канал вывода приходят спайки.

Пример исполнения 20 шагов нейронной сети:

Python

```
# ...
# Создание списка results, в который будут записываться индексы шагов со спайками.
results = []

for step in range(0, 20):
    # Отправка спайков на шагах 0, 5, 10 и 15 исполнения нейронной сети.
    if step % 5 == 0:
        # Отображение номера шага, на котором канал ввода отправил спайк входной проекции.
        print(f'STEP {step}')
        message = SpikeMessage((in_channel_uid, step), [0])
        endpoint.send_message(message)

    backend._step()
    messages_count = endpoint.receive_all_messages()
    output = endpoint.unload_messages(SpikeMessage, out_channel_uid)
    if len(output):
        # Отображение номера шага, на котором на канал вывода пришел спайк.
        print(f'STEP {step}')
```

[Шаги исполнения нейронной сети](#) [?](#)

Исполнение нейронной сети в примере 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 в примере.

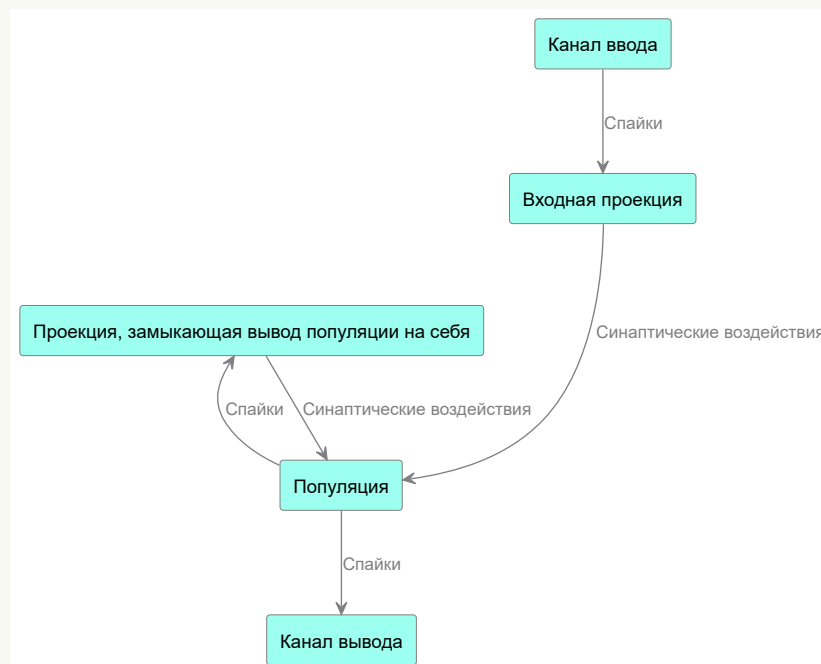


Схема исполнения нейронной сети в примере

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.

13. Запустите программу.

Сценарий исполнения нейронной сети из тернарных слоев для классификации изображений

Вы можете использовать этот сценарий при разработке прикладных решений.

В этом разделе приводится пример создания, обучения, размещения нейронной сети для классификации изображений и ее исполнения на нейроморфном процессоре Алтай-2. Нейронная сеть состоит из [тернарных слоев](#) пакета ANN2SNN.

Сценарий исполнения нейронной сети из тернарных слоев для классификации изображений состоит из следующих этапов:

1 Создание и обучение нейронной сети из тернарных слоев

[Создайте нейронную сеть](#) с помощью тернарных слоев пакета ANN2SNN. В качестве функции активации выходного слоя укажите функцию `softmax`. Подготовьте данные для обучения нейронной сети. Приведите данные для обучения в бинарный вид. Обучите нейронную сеть и оцените результаты обучения.

2 Размещение нейронной сети

Укажите параметры конфигурации нейроморфного процессора Алтай-2 в [YAML-файле](#). Разместите нейронную сеть с помощью следующей команды:

```
placer -p <NN_file> -l <placer_log> -o <altai_config> -hw <hw_config>
```

В этой команде:

- `<NN_file>` – путь к файлу с обученной нейронной сетью;
- `<placer_log>` – путь к файлу, в который будут записываться журналы утилиты `placer`;
- `<altai_config>` – путь к файлу, в который будет сохранена конфигурация для загрузки на нейроморфный процессор;
- `<hw_config>` – путь к файлу, содержащему конфигурацию нейроморфного процессора. Если флаг `-hw` не указан, используется конфигурация нейроморфного процессора по умолчанию.

Пример:

```
placer -p mnist.keras -l mnist_placer.log -o mnist.json
```

3 Запуск инференса нейронной сети

Подключите объединительную плату с нейроморфным процессором Алтай-2 к компьютеру с помощью интерфейсов `PCI-E` или `USB`.

Укажите конфигурационный файл и компонент для исполнения нейронной сети. Нейронная сеть может быть исполнена как на аппаратном процессоре Алтай-2, так и на его программном эмуляторе. Подготовьте данные для инференса и приведите их в бинарный вид. Запустите инференс.

Создание и обучение нейронной сети из тернарных слоев для классификации изображений

Этот раздел содержит инструкции по созданию нейронной сети для классификации изображений и ее обучению. Нейронная сеть создана с помощью тернарных слоев пакета `ANN2SNN`.

Чтобы создать и обучить нейронную сеть из тернарных слоев для классификации изображений:

1. В директории вашего проекта создайте файл программы с расширением `.py`, в котором будет создана и обучена нейронная сеть.
2. В файле программы импортируйте модули библиотек, необходимые для создания и исполнения нейронной сети.

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

Пример подключения объектов для создания и исполнения сверточной нейронной сети:

Python

```
from knp_ann2snn.altainn import TernaryConv2D, TernaryDense
from knp_ann2snn.altainn import heaviside
from knp_ann2snn.altainn import Clip

from tensorflow import keras
from keras.models import Sequential, load_model
from keras.layers import Flatten
from keras.callbacks import ModelCheckpoint

import tensorflow as tf
```

3. Создайте нейронную сеть. Укажите функцию `softmax` в качестве функции активации для выходного слоя.

Пример создания нейронной сети с тремя сверточными слоями:

Python

```
model = Sequential(
    [
        TernaryConv2D(1, 2, strides=2, activation=heaviside, input_shape=(28, 28, 1)),
        Flatten(),
        TernaryDense(10, activation="softmax")
    ]
)
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

4. Подготовьте данные для обучения.

Пример подготовки данных MNIST для обучения:

Python

```
# Загрузка данных MNIST для обучения.
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
# Кодирование целевых меток.
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

# Нормализация входных данных.
x_train = x_train / 255
x_test = x_test / 255
# Приведение входных данных в бинарный вид.
x_train[x_train > 0.5] = 1
x_train[x_train <= 0.5] = 0

x_test[x_test > 0.5] = 1
x_test[x_test <= 0.5] = 0
# Замена размерности данных (28, 28) на (28, 28, 1) для правильной работы сверточных слоев.
x_train = x_train.reshape(x_train.shape + (1,))
x_test = x_test.reshape(x_test.shape + (1,))
```

5. Создайте Callback-функцию для сохранения нейронной сети.

Пример Callback-функции:

Python

```
# Сохранение обученной нейронной сети в файл mnist.h5.
model_checkpoint_callback = ModelCheckpoint(
    filepath="mnist.keras",
    monitor="accuracy",
    mode="max",
    save_best_only=True,
)
```

6. Обучите нейронную сеть.

Пример обучения нейронной сети:

Python

```
# Запуск обучения на тренировочных данных.
model.fit(x_train, y_train, epochs = 15, verbose = 1, callbacks=[model_checkpoint_callback])
# Загрузка обученной модели.
model = load_model(
    "mnist.keras",
    custom_objects = {
        "heaviside_mod": heaviside,
        "TernaryDense": TernaryDense,
        "TernaryConv2D": TernaryConv2D,
        "Clip": Clip
    }
)
# Оценка точности нейронной сети.
print(f"Точность модели: {model.evaluate(x_test, y_test, verbose = 1)[1]*100:.2f}%")
```

7. Запустите программу.

Запуск инференса нейронной сети из тернарных слоев для классификации изображений

Этот раздел содержит инструкции по запуску инференса нейронной сети для классификации изображений на нейроморфном процессоре Алтай-2. Нейронную сеть требуется предварительно [создать](#) с помощью тернарных слоев пакета `ANN2SNN`, обучить и [разместить](#) для исполнения на нейроморфном процессоре.

Чтобы запустить инференс нейронной сети из тернарных слоев для классификации изображений:

1. Подключите объединительную плату с нейроморфным процессором Алтай-2 к компьютеру с помощью интерфейсов `PCI-E` или `USB`.
2. В директории вашего проекта создайте файл программы с расширением `.py`, в котором будет реализован инференс нейронной сети.
3. В файле программы импортируйте модули библиотек, необходимые для инференса нейронной сети. Если требуется, определите псевдонимы с помощью оператора `as`.

Пример подключения объектов для инференса нейронной сети:

Python

```
from knp_ann2snn import Altai
import tensorflow as tf
import numpy as np
from tqdm import tqdm
```

4. Создайте функцию для подсчета точности модели.

Пример функции для подсчета точности модели:

Python

```
def calculate_accuracy(array1,array2):

    correct_predictions = sum(a == b for a, b in zip(array1,array2))
    accuracy = correct_predictions / len(array1)

    return accuracy
```

5. Подготовьте данные для инференса.

Пример подготовки данных MNIST для инференса:

Python

```
# Загрузка модели.
altai = Altai()
# Укажите путь к конфигурационному файлу для загрузки на нейроморфный процессор
# и компонент для исполнения нейронной сети:
# hw - для инференса нейронной сети на аппаратном процессоре Алтай-2
# gm - для инференса нейронной сети на программном эмуляторе.
# Если функция активации на последнем слое - softmax, необходимо ее указать.
altai.build("mnist.json", "gm", activation="softmax")

# Загрузка данных MNIST для инференса.
(_, _), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Нормализация входных данных.
x_test = x_test / 255

# Приведение входных данных в бинарный вид.
x_test[x_test > 0.5] = 1
x_test[x_test <= 0.5] = 0

x_test = x_test.reshape(x_test.shape + (1,))
```

6. Запустите инференс.

Пример инференса:

Python

```
pred = []

for i in tqdm(range(x_test.shape[0])):
    # Преобразование входного вектора в спайки.
    altai.prepare_spikes(x_test[i])
    # Запуск генерации сигнала тик.
    altai.start_ticks(2)
    # Получение спайков от нейронной сети.
    spikes_idx = altai.get_spikes()
    # Берем -1 индекс из-за того, что интересует результат последнего тика.
    predicted_class = spikes_idx[-1]
    pred.append(predicted_class)

# Оценка точности нейронной сети.
print("Accuracy: ", calculate_accuracy(y_test, pred)* 100,"%")
```

7. Запустите программу.

Сценарий исполнения регрессионной модели с обучаемыми блоками, кодирующие и декодирующие данные

Вы можете использовать этот сценарий при разработке прикладных решений.

В этом разделе приводится пример создания, обучения, размещения регрессионной модели и ее исполнения на нейроморфном процессоре Алтай-2.

Сценарий исполнения регрессионной модели с обучаемыми блоками, кодирующие и декодирующие данные, состоит из следующих этапов:

1 Создание и обучение регрессионной модели

[Создайте регрессионную модель](#), состоящую из кодирующего блока, нейронной сети и декодирующего блока. Подготовьте данные для обучения модели. Обучите модель и оцените результаты обучения.

2 Размещение нейронной сети

Укажите параметры конфигурации нейроморфного процессора Алтай-2 в [YAML-файле](#). Разместите нейронную сеть с помощью следующей команды:

```
placer -p <NN_file> -l <placer_log> -o <altai_config> -hw <hw_config>
```

В этой команде:

- `<NN_file>` – путь к файлу с обученной нейронной сетью;
- `<placer_log>` – путь к файлу, в который будут записываться журналы утилиты `placer`;
- `<altai_config>` – путь к файлу, в который будет сохранена конфигурация для загрузки на нейроморфный процессор;
- `<hw_config>` – путь к файлу, содержащему конфигурацию нейроморфного процессора. Если флаг `-hw` не указан, используется конфигурация нейроморфного процессора по умолчанию.

Пример:

```
placer -p snn_sin.keras -l sin_placer.log -o sin.json
```

3 Запуск инференса регрессионной модели

Подключите объединительную плату с нейроморфным процессором Алтай-2 к компьютеру с помощью интерфейсов `PCI-E` или `USB`.

[Укажите](#) конфигурационный файл и компонент для исполнения нейронной сети. Нейронная сеть может быть исполнена как на аппаратном процессоре Алтай-2, так и на его программном эмуляторе. Запустите инференс регрессионной модели.

Создание и обучение регрессионной модели

Этот раздел содержит инструкции по созданию регрессионной модели с обучаемыми блоками, кодирующие и декодирующие данные, и ее обучению. Модель создана с помощью тернарных слоев пакета `ANN2SNN`.

Вы можете использовать эту инструкцию при разработке прикладных решений.

Чтобы создать и обучить регрессионную модель:

1. В директории вашего проекта создайте файл программы с расширением `.py`, в котором будет создана и обучена модель.
2. В файле программы импортируйте модули библиотек, необходимые для создания и обучения модели. Если требуется, определите псевдонимы с помощью оператора `as`.

Пример подключения объектов для создания и обучения регрессионной модели:

Python

```
from knp_ann2snn.altainn import TernaryDense
from knp_ann2snn.altainn import heaviside
from knp_ann2snn.altainn import Clip

from tensorflow import keras
from keras.models import Sequential, load_model
from keras.callbacks import ModelCheckpoint
from keras.layers import Dense

import matplotlib.pyplot as plt
import numpy as np
```

3. Создайте синусоиду и подготовьте данные для обучения.

Пример подготовки данных:

Python

```
# Определение количества элементов в наборе данных.
N = 4000
# Определение индекса, до которого элементы набора данных будут использоваться для обучения.
train_idx = 3000
# Определение индекса, до которого элементы, начиная с индекса train_idx, образуют тестовые данные.
test_idx = 4000
# Определение количества элементов в пакете (англ. batch).
elements_in_batch = 10
# Определение размерности выхода кодирующего блока.
encoder_output_shape = 25
# Определение размерности выхода нейронной сети.
snn_output_shape = 35
# Определение размерности выхода декодирующего блока.
decoder_output_shape = 1

# Создание синусоиды.
x = np.arange(N)
arr = np.sin(x)

# Формирование тренировочных данных на основе elements_in_batch значений.
train_range = range(train_idx - elements_in_batch - 1)
x_train = np.array([arr[i : i + elements_in_batch] for i in train_range])
y_train = np.array([arr[i + elements_in_batch + 1] for i in train_range])

# Формирование тестовых данных на основе elements_in_batch значений.
test_range = range(train_idx, test_idx - elements_in_batch - 1)
x_test = np.array([arr[i : i + elements_in_batch] for i in test_range])
y_test = np.array([arr[i + elements_in_batch + 1] for i in test_range])
```

4. Создайте регрессионную модель. Для этого создайте кодирующий блок, импульсную нейронную сеть и декодирующий блок.

Пример создания регрессионной модели:

Python

```
# Создание кодирующего блока с одним слоем, который преобразует данные в спайки.
encoder = Sequential(
    [
        Dense(
            encoder_output_shape,
            activation=heaviside,
            input_shape=(elements_in_batch,)
        )
    ]
)

# Создание импульсной нейронной сети.
snn = Sequential(
    [
        TernaryDense(
            snn_output_shape,
            activation=heaviside,
            input_shape=(encoder_output_shape,),
            use_bias=False
        )
    ]
)

# Создание декодирующего блока с одним слоем, который преобразует спайки в целевое значение.
decoder = Sequential(
    [
        Dense(
            decoder_output_shape,
            activation="linear",
            input_shape=(snn_output_shape,)
        )
    ]
)

# Создание модели с кодирующим блоком, нейронной сетью и декодирующим блоком.
model = Sequential([encoder, snn, decoder])
model.compile(optimizer="adam", loss="mse")
```

5. Обучите модель.

Пример обучения модели:

Python

```
# Обучение модели на тренировочных данных.
checkpoint = ModelCheckpoint(
    "sin_model.keras",
    monitor="val_loss",
    verbose=1,
    save_best_only=True,
    mode="min"
)
model.fit(x_train, y_train, epochs=50, validation_data=(x_test, y_test), callbacks=[])

model.save("sin_model.keras")

# Загрузка модели.
model = load_model(
    "sin_model.keras",
    custom_objects={
        "heaviside_mod": heaviside,
        "Clip": Clip,
        "TernaryDense": TernaryDense,
    },
)

# Получение предсказаний.
predict_test = model.predict(x_test)
predict_test = predict_test.flatten()

# Построение графиков по предсказанным и целевым значениям.
plt.plot(predict_test, "b")
plt.plot(y_test, "r", alpha=0.5)
plt.savefig("sin_train.png")
```

6. Сохраните кодирующий блок, нейронную сеть и декодирующий блок.

Пример сохранения:

Python

```
# Сохранение кодирующего блока в файл encoder_sin.keras.
model.layers[0].save("encoder_sin.keras")
# Сохранение нейронной сети в файл snn_sin.keras.
model.layers[1].save("snn_sin.keras")
# Сохранение декодирующего блока в файл decoder_sin.keras.
model.layers[2].save("decoder_sin.keras")
```

7. Запустите программу.

Запуск инференса регрессионной модели

Этот раздел содержит инструкции по запуску инференса регрессионной модели с кодирующим и декодирующим блоком. Модель требуется предварительно [создать](#), обучить и [разместить](#) для исполнения на нейроморфном процессоре.

Чтобы запустить инференс регрессионной модели:

1. Подключите объединительную плату с нейроморфным процессором Алтай-2 к компьютеру с помощью интерфейсов PCI-E или USB.
2. В директории вашего проекта создайте файл программы с расширением `.py`, в котором будет реализован инференс модели.

3. В файле программы импортируйте модули библиотек, необходимые для инференса модели

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

Пример подключения объектов для инференса модели:

Python

```
from knp_ann2snn.altainn import TernaryDense
from knp_ann2snn.altainn import heaviside
from knp_ann2snn.altainn import Clip
from knp_ann2snn.python_altai import Altai

from keras.models import load_model
import matplotlib.pyplot as plt

from tqdm import tqdm
import numpy as np
```

4. Создайте синусоиду и подготовьте данные для инференса.

Пример подготовки данных:

Python

```
# Определение количества элементов в наборе данных.
N = 4000
# Определение индекса, до которого элементы набора данных будут использоваться для обучения.
train_idx = 3000
# Определение индекса, до которого элементы, начиная с индекса train_idx, образуют тестовые данные.
test_idx = 4000
# Определение количества элементов в пакете (англ. batch).
elements_in_batch = 10
# Определение размерности выхода кодирующего блока.
encoder_output_shape = 25
# Определение размерности выхода нейронной сети.
snn_output_shape = 35
# Определение размерности выхода декодирующего блока.
decoder_output_shape = 1

# Создание синусоиды.
x = np.arange(N)
arr = np.sin(x)

# Формирование тестовых данных на основе elements_in_batch значений.
test_range = range(train_idx, test_idx - elements_in_batch - 1)
x_test = np.array([arr[i : i + elements_in_batch] for i in test_range])
y_test = np.array([[arr[i + elements_in_batch + 1]] for i in test_range])
```

5. Загрузите модель, кодирующий и декодирующий блок.

Пример загрузки модели, кодирующего и декодирующего блока:

Python

```
altai = Altai()
# Укажите путь к конфигурационному файлу для загрузки на нейроморфный процессор
# и компонент для исполнения нейронной сети:
# hw - для инференса нейронной сети на аппаратном процессоре Алтай-2
# gm - для инференса нейронной сети на программном эмуляторе.
altai.build("sin.json", "gm")

encoder = load_model(
    "encoder_sin.keras",
    custom_objects={
        "heaviside_mod": heaviside
    }
)

snn_model = load_model(
    "snn_sin.keras",
    custom_objects={
        "TernaryDense": TernaryDense,
        "heaviside_mod": heaviside,
        "Clip": Clip
    }
)

decoder = load_model("decoder_sin.keras")
```

6. Запустите инференс.

Пример запуска инференса:

Python

```
predict = np.zeros(len(x_test))

# Запуск цикла предсказаний.
for i in tqdm(range(len(x_test))):
    # Предсказание кодирующего блока.
    encoded = encoder.predict_on_batch(x_test[i:i+1])

    # Предсказание нейронной сети.
    altai.prepare_spikes(encoded)
    altai.start_ticks(1)

    spikes_idx = altai.get_spikes()
    spikes_idx = spikes_idx[spikes_idx!=2147483647]
    spikes = np.zeros(snn_output_shape)
    spikes[spikes_idx] = 1

    # Предсказание декодирующего блока.
    decoded = decoder.predict_on_batch(spikes.reshape(1,-1))

    # Обновление массива предсказаний.
    predict[i] = decoded
    predict = predict.reshape(len(predict))
    y_test = y_test.reshape(len(y_test))
```

7. Оцените результаты инференса. Для этого вы можете сравнить значения синусоиды с предсказаниями модели на графике.

Пример построения графика для оценки результатов инференса:

Python

```
# Построение графиков.  
plt.plot(predict, "b")  
plt.plot(y_test, "r", alpha=0.5)  
plt.savefig("sin_inference.png")  
plt.show()
```

8. Запустите программу.

Использование Kaspersky Neuromorphic Platform API

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

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

- получать данные об используемых устройствах и бэкендах;
- получать данные об объектах библиотеки `core`;
- получать данные об объектах фронтенда платформы;
- управлять популяциями нейронов и связями между ними;
- создавать и обучать нейронные сети;
- управлять гиперпараметрами нейронных сетей;
- создавать новые модели нейронов;
- создавать новые топологии нейронных сетей;
- реализовывать прикладные решения.



[ОТКРЫТЬ СПРАВОЧНИК API для C++ \(на английском языке\)](#)

Участие в разработке платформы

Вы можете участвовать в разработке Kaspersky Neuromorphic Platform. Для этого необходимо получить исходный код из репозитория платформы.

Чтобы участвовать в разработке платформы:

1. Откройте терминал и с помощью команды `cd` перейдите в рабочую директорию.

Вы можете использовать любой другой инструмент для работы с Git. Например, в ОС Windows вы можете использовать [Visual Studio](#).

2. Для получения исходного кода платформы требуется выполнить следующие действия:

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

```
git clone <репозиторий_платформы>
```

где `<репозиторий_платформы>` соответствует веб-адресу репозитория платформы, полученному от сотрудников "Лаборатории Касперского".

- b. Установите Python-пакеты с помощью следующей команды:

```
pip3 -r requirements.txt
```

- c. Если требуется, загрузите актуальную версию удаленного репозитория платформы в ваш локальный репозиторий. Для этого вы можете использовать следующую команду:

```
git pull
```

3. В командной строке выполните следующую команду для создания ветки, в которой вы будете вносить изменения в код платформы:

```
git checkout -b <название_ветки>
```

4. Внесите изменения в код платформы.

5. Зафиксируйте изменения.

Например, вы можете использовать следующие команды для публикации изменений:

```
git add <имя_файла_для_публикации>  
git commit -m "<сообщение_для_публикации>"  
git push origin <название_ветки>
```

6. Опубликуйте изменения в репозитории платформы.

Лицензирование платформы

Условия использования Kaspersky Neuromorphic Platform

Условия использования Kaspersky Neuromorphic Platform – это обязательное соглашение между вами и АО "Лаборатория Касперского", в котором изложены условия, на которых вы можете пользоваться платформой. Kaspersky Neuromorphic Platform распространяется на условиях лицензии Apache 2.0.

Условия использования Kaspersky Neuromorphic Platform распространяются на [исходный код платформы](#), а также следующие пакеты:

- `knp-cpu-single-threaded-backend_<knp_version>_amd64.deb` – deb-пакет с динамической библиотекой однопоточного бэкенда для CPU.
- `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb` – deb-пакет с динамической библиотекой многопоточного бэкенда для CPU.
- `knp-gpu-cuda-backend_<knp_version>_amd64.deb` – deb-пакет с библиотекой бэкенда для GPU с поддержкой технологии CUDA.
- `knp-cpp-framework_<knp_version>_amd64.deb` – deb-пакет с динамической библиотекой фреймворка для C++.
- `knp-cpu-multi-threaded-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb`.
- `knp-cpu-single-threaded-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb`.
- `knp-gpu-cuda-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-gpu-cuda-backend_<knp_version>_amd64.deb`.
- `knp-cpp-framework-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpp-framework_<knp_version>_amd64.deb`.
- `knp-python3-framework_<knp_version>_amd64.deb` – deb-пакет с Python-фреймворком для работы на бэкенде AltAI SNN.
- `knp-examples_<knp_version>_amd64.deb` – deb-пакет с примерами использования Kaspersky Neuromorphic Platform.
- `knp-documentation_<knp_version>_all.deb` – deb-пакет со справочником API.
- `knp-<knp_version>-py3-none-any.whl` – whl-пакет с Python-фреймворком для работы на бэкенде AltAI SNN.

где `<knp_version>` – версия Kaspersky Neuromorphic Platform.

Вы можете ознакомиться с Условиями использования Kaspersky Neuromorphic Platform, прочитав документ `LICENSE.txt`, расположенный в следующих директориях:

- В корневой директории репозитория платформы.
- В директории `/usr/share/doc/knp` после установки пакета `knp-cpp-framework_<knp_version>_amd64.deb`.
- В директории `/usr/lib/python<Python version>/knp` после установки пакета `knp-<knp_version>-py3-none-any.whl`.
- В корневой директории Docker-образа `knp-build-image` для сборки платформы или прикладного решения.

где:

- `<Python version>` – версия установленного Python;
- `<knp_version>` – версия Kaspersky Neuromorphic Platform.

Условия использования бэкендов для нейроморфного процессора Алтай-2 и пакета ANN2SNN

Условия использования бэкендов для [нейроморфного процессора Алтай-2](#) и пакета [ANN2SNN](#) – это обязательное соглашение между вами и ООО "Мотив HT", в котором изложены условия, на которых вы можете пользоваться следующими пакетами:

- `knp-altai-backend_<motiv_version>_amd64.deb` – deb-пакет с динамической библиотекой бэкенда AltAI SNN для нейроморфного процессора Алтай-2;
- `knp-altai-backend-dev_<motiv_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-altai-backend_<motiv_version>_amd64.deb`;
- `knp_ann2snn.deb` – deb-пакет с пакетом ANN2SNN и динамической библиотекой бэкенда AltAI ANN2SNN;
- `knp_ann2snn_x86_64-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом ANN2SNN и динамической библиотекой бэкенда AltAI ANN2SNN для установки на компьютер с процессором архитектуры x86;
- `knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом ANN2SNN и динамической библиотекой бэкенда AltAI ANN2SNN для установки на компьютер с процессором архитектуры ARM.

где `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-2 и пакета ANN2SNN.

Вы можете ознакомиться с Условиями использования следующими способами:

- Во время установки deb-пакетов.
- В документе `LICENSE.txt`, расположенном в директории `/usr/share/doc/altai-backend/`, после установки пакетов `knp-altai-backend_<motiv_version>_amd64.deb` или `knp-altai-backend_<motiv_version>_amd64.deb`.

- В документе `LICENSE.txt`, расположенном в директории `/usr/share/doc/knp_ann2snn/`, после установки пакета `knp_ann2snn.deb`.
- В документе `LICENSE.txt`, расположенном в директории `/usr/lib/python<Python version>/site-packages/knp_ann2snn/`, после установки любого из whl-пакетов.

где:

- `<Python version>` – версия установленного Python;
- `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-2 и пакета `ANN2SNN`.

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

Если вы не согласны с Условиями использования, вы должны удалить и не использовать Kaspersky Neuromorphic Platform. Если вы не принимаете Условия использования бэкендов для нейроморфного процессора Алтай-2 и пакета `ANN2SNN` во время установки deb-пакетов установка пакетов будет прервана.

О предоставлении данных

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

Устранение неисправностей

Этот раздел содержит описание возможных неисправностей в работе Kaspersky Neuromorphic Platform и способов их устранения.

Не выполняется сборка платформы с Python-фреймворком

Проблема

Не выполняется [сборка платформы с Python-фреймворком](#).

Решение

При отсутствии инструмента stubgen на устройстве для [параметра конфигурации](#) KNP_PYTHON_FRAMEWORK_BUILD автоматически задается значение OFF, при котором сборка платформы выполняется без Python-фреймворка. При конфигурации сборки задайте для параметра KNP_PYTHON_FRAMEWORK_BUILD значение ON.

Компилятор или система сборки не может найти библиотеку Boost в ОС Windows

Проблема

Компилятор или система сборки не может найти библиотеку Boost в ОС Windows. При этом путь к библиотеке [указан в параметре Boost_ROOT в файле CMakePresets.json](#).

Решение

Компилятор или система сборки может использовать старое значение параметра Boost_ROOT, хранимое в кеше. В таком случае вам нужно удалить старое значение параметра из кеша. Для этого в Visual Studio на вкладке **Проект** выполните одно из следующих действий:

- Выберите пункт **Удалить кэш и выполнить перенастройку**. Старое значение параметра будет удалено из кеша.
- Выберите пункт **Просмотреть CMakeCache.txt** и удалите значение параметра Boost_ROOT.

После удаления старого значения параметра из кеша повторно [выполните конфигурацию сборки](#).

Приложения

Этот раздел содержит информацию, которая дополняет основной текст документа.

Структура файла, описывающего группы нейронов

Ниже приведен пример структуры HDF5-файла, описывающего группы нейронов:

```
<имя_файла>.h5
├── nodes
│   ├── <название_популяции>
│   │   ├── node_type_id
│   │   ├── node_id
│   │   ├── node_group_id
│   │   ├── node_group_index
│   │   ├── <группа_нейронов>
│   │   │   ├── <атрибут_группы>
│   │   │   ├── ...
│   │   └── dynamics_params
│   │       ├── <динамический_атрибут>
│   │       └── ...
```

В таблице ниже приведены элементы данных в файле, описывающем группы нейронов.

Элементы данных файла, описывающего группы нейронов

Элемент данных	Описание
nodes	Обязательный элемент для определения групп.
<название_популяции>	Обязательный элемент, содержащий название популяции. Популяций может быть несколько.
node_type_id	Обязательный элемент, содержащий набор данных с массивом идентификаторов типов нейронов популяции.
node_id	Обязательный элемент, содержащий набор данных с индексами нейронов популяции.
node_group_id	Обязательный элемент, содержащий набор данных с индексами групп, к которым относятся нейроны популяции.
node_group_index	Обязательный элемент, содержащий набор данных с индексами нейронов в соответствующих группах.
<группа_нейронов>	Обязательный элемент, содержащий идентификатор группы нейронов.
<атрибут_группы>	Один или несколько атрибутов группы. Атрибуты не являются обязательными и зависят от типа нейрона. Каждый из атрибутов представляет собой набор данных.
dynamics_params	Группа необязательных динамических атрибутов.
<динамический_атрибут>	Один или несколько динамических атрибутов. Каждый из атрибутов представляет собой набор данных со значениями атрибутов для каждого нейрона в группе.

Структура файла, описывающего группы синапсов

Ниже приведен пример структуры HDF5-файла, описывающего группы синапсов:

```
<имя_файла>.h5
├── edges
│   ├── <название_проекции>
│   │   └── source_node_id
```

```

| edge_group_id
| edge_group_index
| target_node_id
| edge_type_id
| indices
|   | source_to_target
|   |   | node_id_to_range
|   |   | range_to_edge_id
|   | target_to_source
|   |   | node_id_to_range
|   |   | range_to_edge_id
| <группа_синапсов>
|   | delay
|   | syn_weight
|   | dynamics_params

```

В таблице ниже приведены элементы данных в файле, описывающем группы синапсов.

Элементы данных файла, описывающего группы синапсов

Элемент данных	Описание
edges	Обязательный элемент для определения групп.
<название_проекции>	Обязательный элемент, содержащий название проекции. Проекций может быть несколько.
source_node_id	Обязательный элемент, содержащий набор данных с идентификаторами пресинаптических нейронов для каждого синапса проекции.
edge_group_id	Обязательный элемент, содержащий набор данных с идентификаторами групп, к которым относятся синапсы проекции.
edge_group_index	Обязательный элемент, содержащий набор данных с индексами синапсов в соответствующих группах.
target_node_id	Обязательный элемент, содержащий набор данных с идентификаторами постсинаптических нейронов для каждого синапса проекции.
edge_type_id	Обязательный элемент, содержащий набор данных с идентификаторами типа каждого синапса проекции.
indices	Необязательный элемент для индексации синапсов по пресинаптическим и постсинаптическим нейронам.
source_to_target	Элемент, описывающий синапсы по пресинаптическим нейронам.
target_to_source	Элемент, описывающий синапсы по постсинаптическим нейронам.
node_id_to_range	Элемент, содержащий набор данных с идентификаторами пресинаптических нейронов для каждого синапса проекции.
range_to_edge_id	Элемент, содержащий набор данных с идентификаторами постсинаптических нейронов для каждого синапса проекции.
<группа_синапсов>	Обязательный элемент, содержащий идентификатор группы нейронов.
delay	Необязательный элемент, содержащий набор данных со значениями задержек синапсов группы.
syn_weight	Необязательный элемент, содержащий набор данных со значениями весов синапсов группы.
dynamics_params	Необязательные динамические атрибуты.

Структура и пример файла базового формата

Файл базового формата, описывающий входные или выходные данные, содержит следующие параметры:

- `attributes` – массив атрибутов файла базового формата, содержащий множество пар типа "ключ-значение" со следующими ключами:
 - `name` – название атрибута файла. Предусмотрены следующие названия атрибута: `magic` и `version`. Атрибут `magic` соответствует числу, идентифицирующему формат файла. Атрибут `version` соответствует версии формата.
 - `shape` – размер массива, заданного в параметре `value` для атрибута `version`.
 - `type` – множество пар "ключ-значение", описывающих значение атрибута файла с помощью следующих ключей:
 - `class` – тип значения атрибута файла.
 - `size` – возможный диапазон значения атрибута файла.
 - `endianness` – порядок хранения байтов значения атрибута файла.
 - `value` – значение атрибута файла.
- `spikes` – параметр, описывающий сгенерированные спайки с помощью следующих параметров:
 - `attributes` – массив атрибутов, описывающих способ сортировки сгенерированных спайков, содержащий множество пар "ключ-значение" со следующими ключами:
 - `name` – название атрибута сортировки. Параметр имеет значение `sorting`.
 - `type` – множество пар "ключ-значение", описывающих возможные способы сортировки с помощью следующих ключей:
 - `class` – тип данных, описывающий возможные способы сортировки. Параметр имеет значение `Enumeration`.
 - `mapping` – перечисление возможных способов сортировки спайков и их целочисленные значения. Предусмотрены следующие способы сортировки: `by_id`, `by_time` и `none`. Способ `by_id` соответствует сортировке по индексам нейронов, сгенерировавших спайки. Способ `by_time` соответствует сортировке по временным меткам генерации спайков. Способ `none` соответствует отсутствию сортировки.
 - `value` – выбранный способ сортировки спайков.
 - `node_ids` – индексы нейронов, сгенерировавших спайки. Параметр содержит множество пар "ключ-значение" со следующими ключами:
 - `shape` – размер массива, заданного в параметре `value`.
 - `type` – множество пар "ключ-значение", описывающих нейроны, сгенерировавшие спайки, с помощью следующих ключей:

- `class` – тип значения индексов нейронов.
- `size` – возможный диапазон значений индексов нейронов.
- `endianness` – порядок хранения байтов индексов нейронов.
- `value` – массив индексов UID нейронов, сгенерировавших спайки.
- `timestamps` – временные метки генерации спайков. Параметр содержит множество пар "ключ-значение" со следующими ключами:
 - `attributes` – массив параметров, описывающих временные метки генерации спайков. Массив содержит параметр `values`, соответствующий типу значения временных меток. Параметр `values` имеет значение `step`, что соответствует шагам исполнения нейронной сети. Помимо элемента `values` массив `attributes` содержит множество пар "ключ-значение" со следующими ключами:
 - `name` – название атрибута временной метки. Параметр имеет значение `units`.
 - `type` – множество пар "ключ-значение", описывающих временные метки. Ключ `class` соответствует типу значения, заданного для параметра `values`. Ключ `charSet` соответствует стандарту, используемому для кодирования значений временных меток.
 - `shape` – размер массива, заданного в параметре `value`.
- `type` – множество пар "ключ-значение", описывающих значения временной метки с помощью следующих ключей:
 - `class` – тип значения временных меток.
 - `endianness` – порядок хранения байтов значений временных меток.
- `value` – временные метки генерации спайков.

Ниже приведен пример файла базового формата в нотации JSON.

```
{
  "attributes": [
    {
      "name": "magic",
      "type": {
        "class": "Integer (unsigned)",
        "size": 32,
        "endianness": "little-endian"
      },
      "value": 2682
    },
    {
      "name": "version",
      "shape": [2],
      "type": {
        "class": "Integer (unsigned)",
        "size": 32,
        "endianness": "little-endian"
      },
      "value": [0, 1]
    }
  ],
  "spikes": {
    "attributes": [
      {
        "name": "sorting",
        "type": {
          "class": "Enumeration",
          "mapping": {
```

```

        "by_id": 1,
        "by_time": 2,
        "none": 0
    },
    "value": "by_time"
}
},
],
"node_ids": {
    "shape": [5],
    "type": {
        "class": "Integer (unsigned)",
        "size": 64,
        "endianness": "little-endian"
    },
    "value": [1, 2, 3, 4, 5]
},
"timestamps": {
    "attributes": [
        {
            "name": "units",
            "type": {
                "class": "String",
                "charSet": "ASCII"
            }
        },
        {
            "name": "step",
            "type": {
                "class": "Float",
                "endianness": "little-endian"
            }
        }
    ],
    "value": [1, 2, 3, 4, 5]
}
}
}

```

Результаты оценки качества модели, передаваемые в формате CSV

При вызове метода записи результатов оценки модели класса [InferenceResultsProcessor](#) в поток данных записываются следующие данные в CSV-формате:

- **CLASS** – индекс класса объектов.
- **TOTAL_VOTES** – общее количество голосов для класса. Значение соответствует сумме истинно положительных, ложно положительных и ложно отрицательных результатов.
- **TRUE_POSITIVES** – количество истинно положительных результатов для класса.
- **FALSE_NEGATIVES** – количество ложно отрицательных результатов для класса.
- **FALSE_POSITIVES** – количество ложно положительных результатов для класса.
- **TRUE_NEGATIVES** – количество истинно отрицательных результатов для класса.
- **PRECISION** – точность модели относительно результатов для конкретного класса объектов.
- **RECALL** – полнота модели относительно результатов для конкретного класса объектов.
- **PREVALENCE** – распространенность модели относительно результатов для конкретного класса объектов.

- `ACCURACY` – правильность модели относительно результатов для конкретного класса объектов.
- `F_MEASURE` – F-мера относительно результатов для конкретного класса объектов.

Формулы для расчета метрик оценки качества модели

С помощью функций пространства имен `inference_evaluation` вы можете получить следующие метрики оценки качества модели:

- Точность.
- Полнота.
- Распространенность.
- Правильность.
- F-мера.

Точность

Метрика вычисляется по формуле, представленной на рисунке ниже, где:

- `Precision` соответствует метрике точности.
- `TP` соответствует количеству истинно положительных результатов.
- `FP` соответствует количеству ложно положительных результатов.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Формула для расчета метрики точности

Полнота

Метрика вычисляется по формуле, представленной на рисунке ниже, где:

- `Recall` соответствует метрике полноты.
- `TP` соответствует количеству истинно положительных результатов.
- `FN` соответствует количеству ложно отрицательных результатов.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Формула для расчета метрики полноты

Распространенность

Метрика вычисляется по формуле, представленной на рисунке ниже, где:

- Prevalence соответствует метрике распространенности.
- TP соответствует количеству истинно положительных результатов.
- FN соответствует количеству ложно отрицательных результатов.
- total соответствует общему количеству результатов.

$$\text{Prevalence} = \frac{\text{TP} + \text{FN}}{\text{total}}$$

Формула для расчета метрики распространенности

Правильность

Метрика вычисляется по формуле, представленной на рисунке ниже, где:

- Accuracy соответствует метрике правильности.
- TP соответствует количеству истинно положительных результатов.
- TN соответствует количеству истинно отрицательных результатов.
- total соответствует общему количеству результатов.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{total}}$$

Формула для расчета метрики правильности

F-мера

Метрика вычисляется по формуле, представленной на рисунке ниже, где:

- F-score соответствует метрике F-меры.
- Precision соответствует метрике точности.
- Recall соответствует метрике полноты.

$$\text{F-score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Формула для расчета метрики F-меры

Параметры и пример YAML-файла, содержащего конфигурацию нейроморфного процессора Алтай-2

Параметры конфигурационного файла необходимо задавать для исполнения нейронной сети на бэкенде AltAI ANN2SNN.

В конфигурационном файле задается информация о количестве СБИС, ядер, нейронов и синапсов в нейроморфном процессоре Алтай-2. Конфигурационный файл содержит следующие параметры:

- `chip` – параметр, включающий параметр `cores`. В параметре `cores` задается количество ядер в одной СБИС.
- `core` – группа параметров, описывающих ядро с помощью следующих параметров:
 - `neurons` – количество нейронов в ядре.
 - `axons` – количество входных линий.
 - `synapses` – количество синапсов в ядре.
- `row` – параметр, включающий параметр `chips`. В параметре `chips` задается количество СБИС в одном модуле объединительной платы нейроморфного процессора Алтай-2.
- `col` – параметр, включающий параметр `chips`. В параметре `chips` задается количество модулей в объединительной плате нейроморфном процессоре Алтай-2.
- `not_working_chips` – массив с номерам СБИС, которые не будут участвовать в исполнении нейронной сети на нейроморфном процессоре, например, из-за их отсутствия на модулях объединительной платы.

Этот параметр заполняется квалифицированным техническим специалистом Заказчика, сотрудником "Лаборатории Касперского" или сертифицированным интегратором.

Ниже приведен пример YAML-файла, содержащего конфигурацию нейроморфного процессора Алтай-2 по умолчанию. В примере ниже объединительная плата нейроморфного процессора состоит из 4 модулей по 8 СБИС в каждом модуле. Каждая СБИС содержит 16 ядер, каждое из которых включает в себя по 512 нейронов, синапсов и входных линий.

hw_config.yaml

```
chip:
  cores: 16
core:
  neurons: 512
  axons: 512
  synapses: 512
row:
  chips: 8
col:
  chips: 4
not_working_chips: []
```

Параметры конфигурации сборки, специфичные для Kaspersky Neuromorphic Platform

Помимо стандартных параметров конфигурации CMake, при [конфигурации сборки](#) вы можете использовать параметры, специфичные для Kaspersky Neuromorphic Platform. Эти параметры описаны в таблице ниже.

Параметры конфигурации сборки, специфичные для Kaspersky Neuromorphic Platform

Параметр конфигурации	Возможные значения	Описание
<code>KNP_BUILD_CUDA</code>	Предусмотрены следующие значения: <code>ON</code> – включает сборку deb-пакета с библиотекой бэкенда для GPU с поддержкой технологии CUDA. При наличии программного обеспечения CUDA Toolkit на устройстве параметру автоматически присваивается значение <code>ON</code>. <code>OFF</code> – выключает сборку deb-пакета с библиотекой бэкенда для GPU с поддержкой технологии CUDA. При отсутствии программного обеспечения CUDA Toolkit на устройстве параметру автоматически присваивается значение <code>OFF</code>.	Параметр регулирует сборку deb-пакета с библиотекой бэкенда для GPU с поддержкой технологии CUDA.
<code>KNP_BUILD_DOCUMENTATION</code>	Предусмотрены следующие значения: <code>ON</code> – включает сборку deb-пакета со справочником API. <code>OFF</code> – выключает сборку deb-пакета со справочником API.	Параметр регулирует сборку deb-пакета со справочником API.
<code>KNP_BUILD_EXAMPLES</code>	Предусмотрены следующие значения: <code>ON</code> – включает сборку deb-пакета с примерами использования Kaspersky Neuromorphic Platform. <code>OFF</code> – выключает сборку deb-пакета с примерами использования Kaspersky Neuromorphic Platform.	Параметр регулирует сборку deb-пакета с примерами использования Kaspersky Neuromorphic Platform.
<code>KNP_BUILD_TESTS</code>	Предусмотрены следующие значения: <code>ON</code> – включает сборку тестов. <code>OFF</code> – выключает сборку тестов.	Параметр регулирует сборку тестов.
<code>KNP_ENABLE_AVX</code>	Предусмотрены следующие значения: <code>ON</code> – включает использование расширения Advanced Vector Extensions. <code>OFF</code> – выключает использование расширения Advanced Vector Extensions.	Параметр регулирует использование расширения Advanced Vector Extensions (avx).
<code>KNP_ENABLE_COVERAGE</code>	Предусмотрены следующие значения: <code>ON</code> – включает проверку тестами исходного кода платформы. <code>OFF</code> – выключает проверку тестами исходного кода платформы.	Параметр регулирует проверку покрытия тестами исходного кода платформы.
<code>KNP_IPO_ENABLED</code>	Предусмотрены следующие значения: <code>ON</code> (по умолчанию) – включает оптимизацию во время компоновки. <code>OFF</code> – выключает оптимизацию во время компоновки.	Параметр регулирует, используется ли оптимизация во время компоновки.

KNP_INSTALL	Предусмотрены следующие значения: - ON – включает вызов команды <code>install</code> в CMake. - OFF – выключает вызов команды <code>install</code> в CMake.	Параметр регулирует вызов команды <code>install</code> в CMake.
KNP_MAINTAINER_BUILD	Предусмотрены следующие значения: - ON – включает сборку для специалиста по сопровождению. - OFF – выключает сборку для специалиста по сопровождению.	Параметр регулирует сборку для специалиста по сопровождению платформы (англ. maintainer).
KNP_PYTHON_FRAMEWORK_BUILD	Предусмотрены следующие значения: - ON – включает сборку платформы с Python-фреймворком. При наличии инструмента stubgen на устройстве параметру автоматически присваивается значение ON. - OFF – включает сборку платформы без Python-фреймворка. При отсутствии инструмента stubgen на устройстве параметру автоматически присваивается значение OFF.	Параметр регулирует, будет ли выполнена сборка платформы с Python-фреймворком.
KNP_PYTHON_BUILD_WHEEL	Предусмотрены следующие значения: - ON – включает сборку whl-пакета для Python-фреймворка. - OFF – выключает сборку whl-пакет для Python-фреймворка.	Параметр регулирует, будет ли выполнена сборка whl-пакета для Python-фреймворка. Значение параметра зависит от параметров KNP_MAINTAINER_BUILD и KNP_PYTHON_FRAMEWORK_BUILD.

Пресеты для сборки платформы

В файле `CMakePresets.json`, расположенном в корневой директории с исходным кодом платформы, описаны пресеты, которые вы можете использовать для [сборки платформы](#). Kaspersky Neuromorphic Platform предоставляет следующие пресеты:

linux-default

Используется в качестве пресета по умолчанию при сборке на Linux. Для сборки используется инструмент Ninja.

windows-default

Используется в качестве пресета по умолчанию при сборке на Windows. Для сборки используется Visual Studio 17 2022.

windows-nmake

Используется для сборки на Windows с помощью утилиты NMake.

windows-ninja

Используется для сборки на Windows с помощью инструмента Ninja.

Глоссарий

Artificial Neural Network (ANN)

Искусственная нейронная сеть, работающая послойно, с числами на входах и выходах нейронов слоя.

Spiking Neural Network (SNN)

Искусственная нейронная сеть, работающая на основе асинхронной передачи импульсов (спайков) между нейронами.

Бэкенд

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

Бэкенд AltAI ANN2SNN

Бэкенд для обучения и исполнения искусственных нейронных сетей, преобразованных в импульсные, на нейроморфном процессоре Алтай-2.

Бэкенд AltAI SNN

Бэкенд для исполнения импульсных нейронных сетей на нейроморфном процессоре Алтай-2.

Вес синапса

Значение сигнала, который синапс передает нейронам связанной популяции.

Внешняя среда

Совокупность внешних по отношению к Kaspersky Neuromorphic Platform элементов, с которыми взаимодействует платформа.

Вычислитель

Устройство или код, непосредственно выполняющий сеть.

Механизм Winner Takes ALL (WTA)

Механизм, используемый в алгоритмах конкурентного обучения для выбора наиболее представительного или доминирующего признака или закономерности в наборе данных.

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 спайк 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 другими нейронами через множество синапсов.

Синаптическое воздействие

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

Соединение

Связь синапса с пресинаптическим или постсинаптическим нейроном.

Спайк

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

Такт

Квант времени синхросигнала, за который вычислитель ядра проводит одну операцию.

Тик

Квант времени, за который выполняется вычисление функции всех нейронов в ядре. За время тика все спайки, сгенерированные ядрами или поступившие на входные линии нейронов, доставляются до целевых нейронов.

Фреймворк

Программная часть платформы, определяющая ее структуру и обеспечивающая управление бэкендами. В Kaspersky Neuromorphic Platform реализован фреймворк на языке программирования C++.

Функция нейрона

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

Функция синапса

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

Информация о стороннем коде

Информация о стороннем коде Kaspersky Neuromorphic Platform содержится в файлах `NOTICE.txt`, расположенных в следующих директориях:

- При работе с исходным кодом платформы в корневой директории репозитория платформы.
- При работе с Docker-образом `knp-build-image` в корневой директории Docker-образа.
- После установки пакета `knp-cpp-framework_<knp_version>_amd64.deb` в директории `/usr/share/doc/knp`.
- После установки пакета `knp-<knp_version>-py3-none-any.whl` в директории `/usr/lib/python<Python version>/knp`.

где:

- `<Python version>` – версия установленного Python;
- `<knp_version>` – версия Kaspersky Neuromorphic Platform.

Информация о стороннем коде пакетов, поставляемых ООО "Мотив НТ", содержится в файле `NOTICE.txt`, расположенном в следующих директориях:

- `/usr/share/doc/altai-backend/` после установки пакетов `knp-altai-backend_<motiv_version>_amd64.deb` или `knp-altai-backend-dev_<motiv_version>_amd64.deb`.
- `/usr/share/doc/knp_ann2snn/` после установки пакета `knp_ann2snn.deb`.
- `/lib/python<Python version>/site-packages/knp_ann2snn/` после установки пакетов `knp_ann2snn_x86_64-<motiv_version>-py3-none-any.whl` или `knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl`.

где:

- `<Python version>` – версия установленного Python;
- `<motiv_version>` – версия бэкендов для [нейроморфного процессора Алтай-2](#) и пакета [ANN2SNN](#).

Уведомления о товарных знаках

Зарегистрированные товарные знаки и знаки обслуживания являются собственностью их правообладателей.

AMD и AMD64 – товарные знаки или зарегистрированные товарные знаки Advanced Micro Devices, Inc.

Apache является либо зарегистрированным товарным знаком, либо товарным знаком Apache Software Foundation.

Apple, Leopard, Mac, Mac OS, macOS, OS X, Objective-C, Safari, TrueType и Xcode – товарные знаки Apple Inc.

Arm – зарегистрированный товарный знак Arm Limited (или дочерних компаний) в США и/или других странах.

Borland – товарный знак или зарегистрированный товарный знак Borland Software Corporation.

Ubuntu и LTS являются зарегистрированными товарными знаками Canonical Ltd.

Docker и логотип Docker являются товарными знаками или зарегистрированными товарными знаками компании Docker, Inc. в США и/или других странах. Docker, Inc. и другие стороны могут также иметь права на товарные знаки, описанные другими терминами, используемыми в настоящем документе.

Знак FreeBSD является зарегистрированным товарным знаком фонда FreeBSD.

GITHUB – товарный знак GitHub, Inc., зарегистрированный в США и других странах.

GITLAB – товарный знак GitLab Inc. в США и других странах и регионах.

Google, Chrome и Chromium – товарные знаки Google LLC.

TensorFlow и любые связанные с ним обозначения являются товарными знаками Google LLC.

Intel, Core и Pentium являются товарными знаками Intel Corporation или ее дочерних компаний.

IBM – товарный знак International Business Machines Corporation, зарегистрированный во многих юрисдикциях по всему миру.

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

Microsoft, Consolas, Internet Explorer, OpenType, Segoe, Visual Studio, Win32 и Windows являются товарными знаками группы компаний Microsoft.

Firefox является товарным знаком Mozilla Foundation в США и других странах.

NVIDIA является зарегистрированным товарным знаком NVIDIA Corporation.

Java и JavaScript – зарегистрированные товарные знаки компании Oracle и/или аффилированных компаний.

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

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

QT – товарный знак или зарегистрированный товарный знак The Qt Company Ltd.

UNIX – товарный знак, зарегистрированный в США и других странах, использование лицензировано X/Open Company Limited.