**Инструкция пользователя**

**GOLEM** — open-source фреймворк структурного обучения композтных моделей ИИ. Области применения:

- Автоматическое построение структурированных моделей в форме направленных графов с помощью эволюционных методов оптимизации.

- Визуализация процесса оптимизации и результирующих моделей для повышения интерпретируемости результата человеком.

На рисунке 1 приведена общая структура фреймворка. Она включает в себя блоки, предназначенные для определения предметной области (в качестве примеров приведены предметные области оптимизации графов мат. операций (EPDE) и байесовских сетей (БС), а также графов машинного обучения) и для решения задач создания композитных моделей из указанных элементов предметной области.

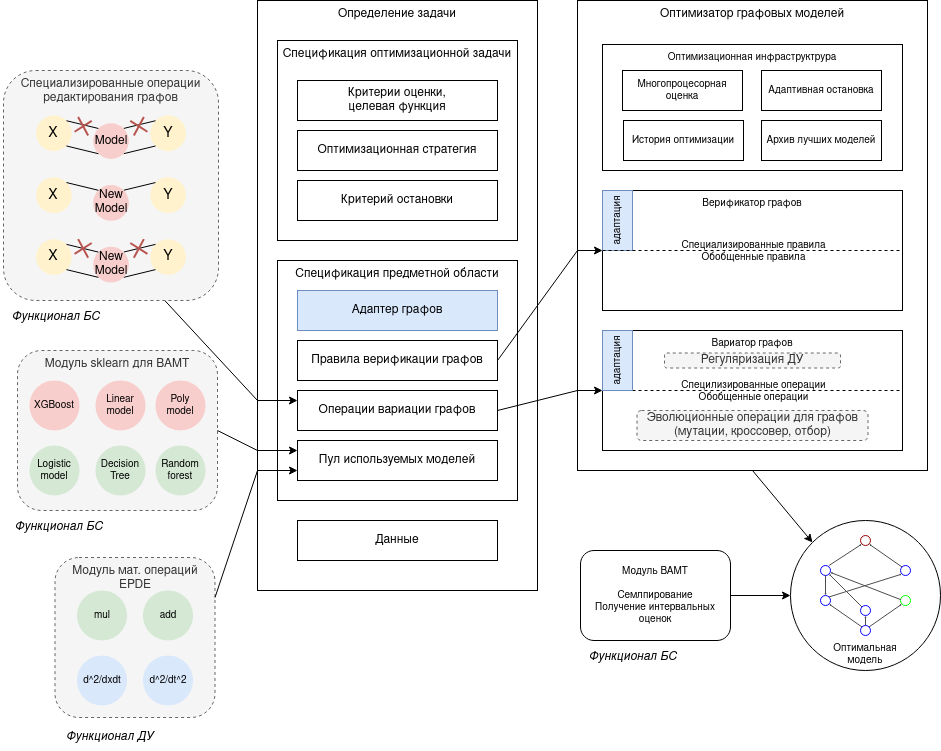


Рисунок 1 – Структура фреймворка

Фреймворк состоит из 2 подсистем: первая обобщает алгоритм на любую предметную область, вторая отвечает непосредственно за алгоритм оптимизации и модулей. Подсистема спецификации под предметную область включает модули, задающие правила верификации моделей, операции вариации графов (например, специализированные эволюционные операторы в случае эволюционного оптимизатора), модуль предметных моделей, из которых будет строится граф (например, операций машинного обучения из библиотеки sklearn или математические операции в случае ДУ), и модуль адаптера предметных графов в универсальное графовое представление, которым оптимизатор манипулирует в ходе работы. Подсистема алгоритма графовой оптимизации включает в себя модули общей оптимизационной инфраструктуры (история оптимизации, архив лучших найденных моделей, логику адаптивной остановки в соответствии с указанными критериями остановки, вычислительный модуль, оценивающий целевую функцию), а также сами верификатор и вариатор графов, который включает как обобщенные правила, так и специализированные для предметной области, задаваемые первой подсистемой (например, операция регуляризации графа ДУ). Адаптер графов позволяет автоматически адаптировать и восстанавливать предметную структуру графа в соответствии со случаем обобщенного либо специализированного правила вариации/верификации. Дополнительные модули предметной области могут включаться в ход оптимизации через оптимизационную инфраструктуру (механизм callback) либо подключаться к результату работы оптимизации.

Предметные области оптимизации байесовских сетей и дифференциальных уравнений используют графовый оптимизатор с различными настройками и подключают дополнительные модули. Для дифференциальных уравнений графовому оптимизатору в качестве моделей для вершин графа передаются математические операции, передается специализированная целевая функция, а обобщенные операции эволюционной вариации графов дополняются специализированной операцией регуляризации ДУ.

**Пример работы с фреймворком для решения задачи классификации временных рядов**

* **Шаг 1**. Инициализация ограничений на графы (списка доступных типов узлов), целевой функции, параметров оптимизации (таймаут).

node\_types = ('a', 'b')

target\_graph = generate\_labeled\_graph('tree', size, node\_types)

objective = Objective(partial(tree\_edit\_dist, target\_graph))

initial\_population = [generate\_labeled\_graph('tree', 5, node\_types)

for \_ in range(10)]

requirements = GraphRequirements(timeout=timedelta(minutes=timeout), n\_jobs=-1)

gen\_params = GraphGenerationParams(adapter=BaseNetworkxAdapter(),

available\_node\_types=node\_types)

algo\_params = GPAlgorithmParameters(pop\_size=30)

* **Шаг 2**. Инициализация оптимизатора и запуск эксперимента.

optimiser = EvoGraphOptimizer(objective, initial\_population,

requirements, gen\_params, algo\_params)

found\_graphs = optimiser.optimise(objective)

* **Шаг 3**. Визуализация результатов эксперимента.

found\_graph = gen\_params.adapter.restore(found\_graphs[0])

draw\_graphs\_subplots(target\_graph, found\_graph,

titles=['Target Graph', 'Found Graph'])

optimiser.history.show.fitness\_line()

return found\_graph

Параметры конфигурации:

* objective – целевая функция, задающая задачу оптимизации
* initial\_population – начальное приближение (поколение индивидов)
* available\_node\_types – доступные типы узлов
* pop\_size – размер популяции для эволюционной оптимизации
* adapter – адаптер структуры предметной области ко внутреннему графовому представлению
* timeout – максимальное количество времени для поиска графа
* n\_jobs – количество процессов для параллельного выполнения

**Компонент размещен в зеркалированных репозиториях:**

* <https://github.com/aimclub/golem>
* <https://gitlab.actcognitive.org/itmo-nss-team/golem>