



ITMO UNIVERSITY

Saint Petersburg, Russia

# Как и зачем создавать custom-решения на основе фреймворка FEDOT

<https://github.com/nccr-itmo/FEDOT>

Никитин Николай,  
к.т.н, старший научный сотрудник НЦКР

# Что такое FEDOT?

FEDOT – open-source фреймворк для решения задач автоматического машинного обучения.

## Основные идеи:

- **Автоматическое создание ML-пайплайнов** произвольной структуры;
- **Структурная оптимизация** с помощью генетического программирования, настройка гиперпараметров с помощью hyperopt;
- Поддержка **различных типов данных** (таблицы, текст, изображения, временные ряды) и соответствующие им задачи – в рамках одного пайплайна;
- **Модульность**, расширяемость, интегрируемость с ML-инструментами;
- Сочетание **легковесного API** для конечного пользователя и расширенного конфигурирования для исследовательских задач.

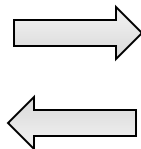
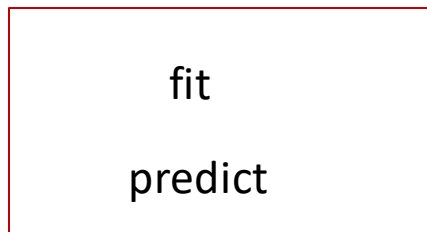
Репозиторий FEDOT: <https://github.com/nccr-itmo/FEDOT>

Исходный код покрыт модульными и интеграционными тестами.

Доступна установка из PyPI. Текущая версия – 0.3.1

# Как работает FEDOT – оптимизация пайплайнов МО

## API



## GPComposer

Блок AutoML для  
пайплайнов МО

Специальные  
эв. операторы:

- Настройка гиперпараметров
- Бустинг

Целевая функция

Классы:

Pipeline, PrimaryNode,  
SecondaryNode, PipelineChangeAdvisor

## GPOptimizer

Эволюционный  
графовый  
оптимизатор  
GPComp@Free

Общие операторы:

- Мутация для графов
- Кроссовер для графов

Классы:

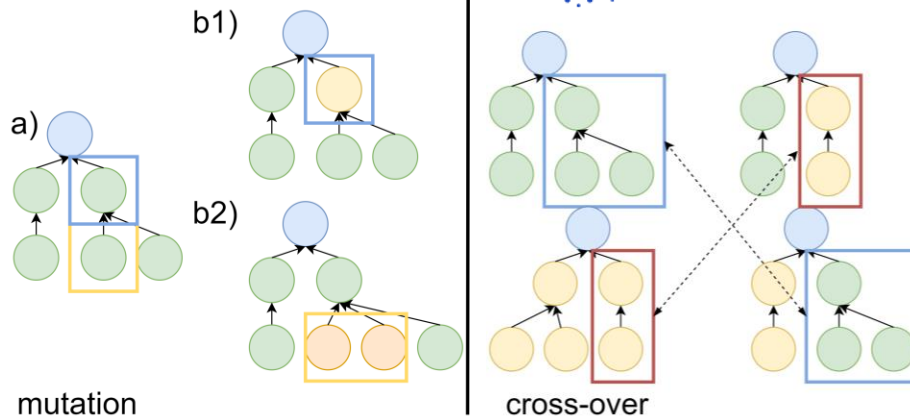
OptGraph, OptNode

PipelineAdapter

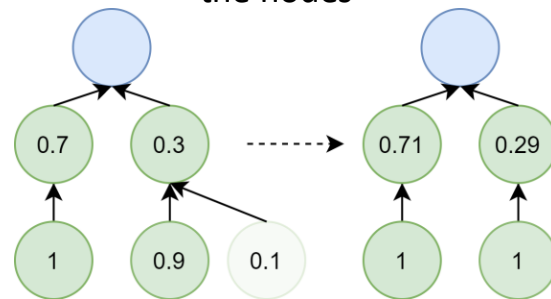
# Какие ещё задачи могут быть решены таким образом?

Из последних разработок:

- FEDOT – строит DAG из моделей МО
- EPDE – строит DAG из членов диф. уравнений и операторов
- EPDE spin-off – строит DAG из элементарных функций и операторов
- BAMT – строит DAG баесовской сети из переменных
- FEDOT-NAS – строит DAG из слоёв сверточной нейросети



Here we don't need to know anything about the nodes

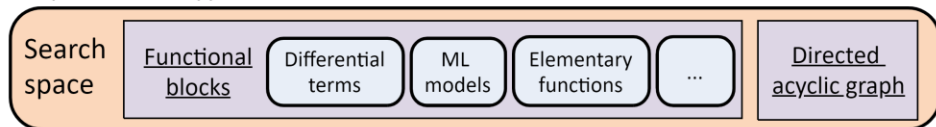


Here we have to invent something for every model type

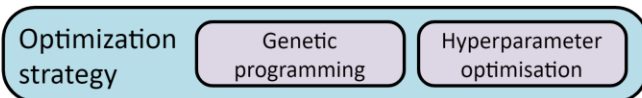
# Что такое FEDOT, если смотреть шире?

Что определяет порядок автоматического создания модели?

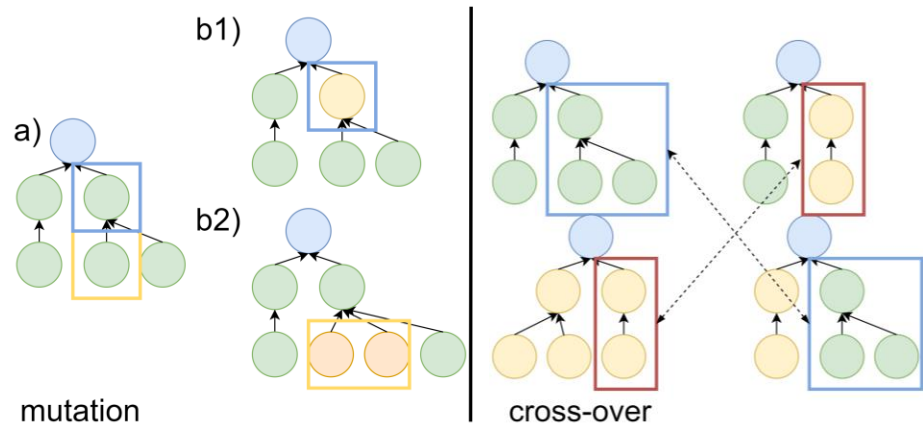
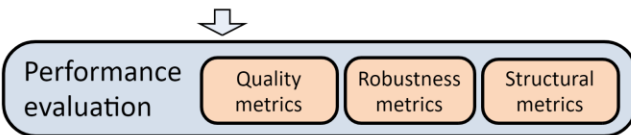
Step 1: which type of model do we want to discover?



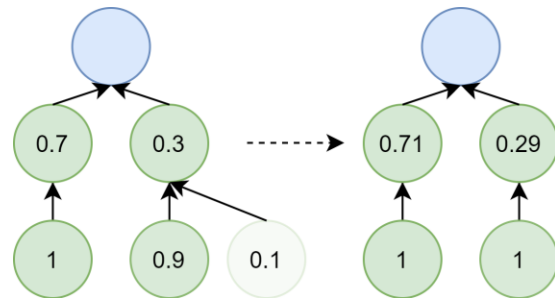
Step 2:  
how to discover  
the model?



Step 3:  
how to analyze  
the properties of  
the model?



Эволюционные операторы обычно не зависят от задачи...



Но есть и специализированные алгоритмы

# Как применить FEDOT для новых задач?

FEDOT – инструмент для  
решения задач  
автоматического  
моделирования

Новая задача

## CustomComposer

Блок  
автоматического  
создания модели

Специальные  
эв. операторы:  
...

Целевая функция

Классы:  
CustomModel,  
CustomNode, Advisor

## CustomAdapter

## GPOptimizer

Эволюционный  
графовый  
оптимизатор  
GPComp@Free

Общие операторы:

- Мутация для графов
- Кроссовер для графов

Классы:  
OptGraph, OptNode

# В чем смысл такого решения?

## Преимущества:

- Объединение опыта разных групп внутри NSS Lab и за её пределами;
- Пере-использование наработок по многокритериальной оптимизации из FEDOT;
- Большая надежность за счет покрытия unit-тестами
- Устранение дублирования эв. алгоритмов в разных инструментах;
- Упрощение экспериментов (есть готовая обвязка);
- Возможность использования единообразной визуализации эволюции и графовых моделей (см <https://github.com/nccr-itmo/FEDOT.Web>);

## Недостатки:

- Нужно разбираться в FEDOT;
- Сложности с управлением зависимостями (т.к. часть логики – внешняя);
- Нужно следить за правилами валидации;

# Возможные сценарии применения

## 1. Neural Architecture Search

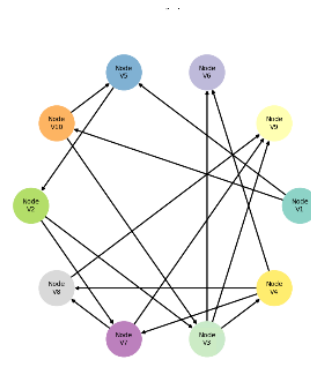
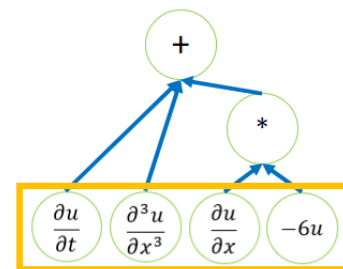
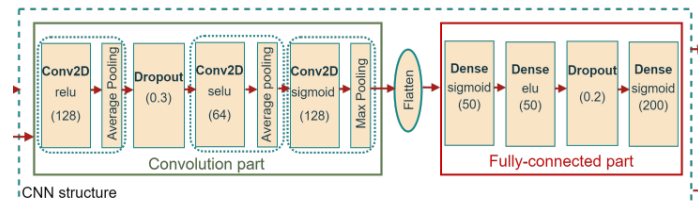
- OptNode – слой CNN
- OptGraph – вся CNN
- Adapter – преобразует в Keras-модель
- Целевая функция – ошибка предсказания

## 2. Equation Discovery

- OptNode – токен (пример -  $dx/dy$ )
- OptGraph – всё уравнение
- Adapter – преобразует в DAG из Equation
- Целевая функция – LASSO + ошибка

## 3. Bayesian Network

- OptNode – переменная
- OptGraph – вся сеть
- Adapter – преобразует в BayesianNetwork
- Целевая функция – K2 score





## Этапы:

1. Определить целевую функцию (на входе – graph, на выходе – List[float])
2. а) Или унаследовать оптимизируемый объект от OptNode и OptGraph  
б) Или создать адаптер, унаследованный от BaseOptimizationAdapter
3. Выбрать операторы мутации и кроссовера, при необходимости – создать новые
4. Выбрать функции структурных ограничений, при необходимости – создать новые
5. Сконфигурировать оптимизатор (число поколений, топологические ограничения и т.д.)
6. Запустить optimizer.optimise

# Пример реализации для «условной» модели (1/3)

```
def custom_metric(graph: CustomGraphModel, data: pd.DataFrame):  
    graph.show()  
    existing_variables_num = -graph.depth - graph.evaluate(data)  
  
    return [existing_variables_num]
```

Новая целевая функция

```
def _has_no_duplicates(graph):  
    _, labels = graph_structure_as_nx_graph(graph)  
    list_of_nodes = [str(node) for node in labels.values()]  
    if len(list_of_nodes) != len(set(list_of_nodes)):  
        raise ValueError('Custom graph has duplicates')  
    return True
```

Новое ограничение

# Пример реализации для «условной» модели (2/3)

```
def custom_mutation(graph: OptGraph, **kwargs):
    num_mut = 10
    try:
        for _ in range(num_mut):
            rid = random.choice(range(len(graph.nodes)))
            random_node = graph.nodes[rid]
            other_random_node = graph.nodes[random.choice(range(len(graph.nodes)))]
            nodes_not_cycling = (random_node.descriptive_id not in
                                [n.descriptive_id for n in other_random_node.ordered_subnodes_hierarchy()] and
                                other_random_node.descriptive_id not in
                                [n.descriptive_id for n in random_node.ordered_subnodes_hierarchy()])
            if nodes_not_cycling:
                graph.operator.connect_nodes(random_node, other_random_node)
    except Exception as ex:
        graph.log.warn(f'Incorrect connection: {ex}')
    return graph
```

Новая  
мутация

Выбор правил валидации и  
начального приближения  
(опционально)

```
rules = [has_no_self_cycled_nodes, has_no_cycle, _has_no_duplicates]
```

```
initial = CustomGraphModel(nodes=[CustomGraphNode(nodes_from=None,
                                                    content=node_type) for node_type in nodes_types])
```

# Пример реализации для «условной» модели (3/3)

```
requirements = GPComposerRequirements(  
    primary=nodes_types,  
    secondary=nodes_types, max_arity=10,  
    max_depth=10, pop_size=5, num_of_generations=5,  
    crossover_prob=0.8, mutation_prob=0.9, timeout=timeout)
```

Параметры ген. алгоритма

```
optimiser_parameters = GPGraphOptimiserParameters(  
    genetic_scheme_type=GeneticSchemeTypeEnum.steady_state,  
    mutation_types=[custom_mutation],  
    crossover_types=[CrossoverTypeEnum.none],  
    regularization_type=RegularizationTypeEnum.none)
```

Настройки ген. схемы и операторов

Настройка адаптеров и ограничений

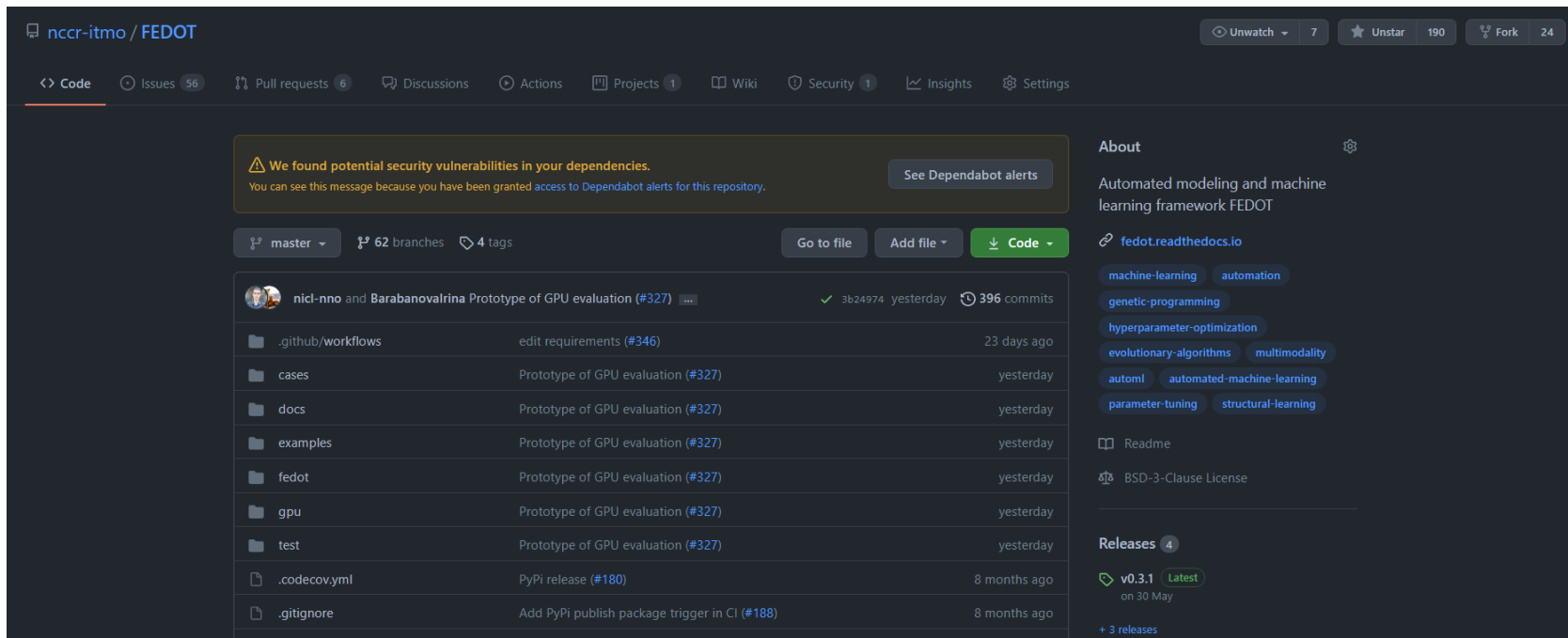
```
graph_generation_params = GraphGenerationParams(  
    adapter=DirectAdapter(base_graph_class=CustomGraphModel, base_node_class=CustomGraphNode),  
    rules_for_constraint=rules)
```

# Как удобно подключить FEDOT к вашему коду?

1. **pip install fedot** – плохой вариант, т.к. работа будет идти не с последней версией;
2. **Скопировать** код из репозитория FEDOT и вставить к себе в проект – плохой вариант, т.к. невозможна синхронизация изменений;
3. **pip install <https://github.com/nccr-itmo/FEDOT/archive/<branchname>.zip> --upgrade --force-reinstall --no-deps --no-cache-dir** – годится, если изменений в ядро нужно не очень много (т.к. после каждого придется делать git commit в <branchname>);
4. **git submodule** – несколько громоздкая настройка, но потом можно сравнительно удобно редактировать код библиотек, от которых зависит основной проект.

# Как внести свои изменения в FEDOT

1. Создать fork, внести изменения в оптимизатор
2. Покрыть их тестами
3. Предложить pull request в master

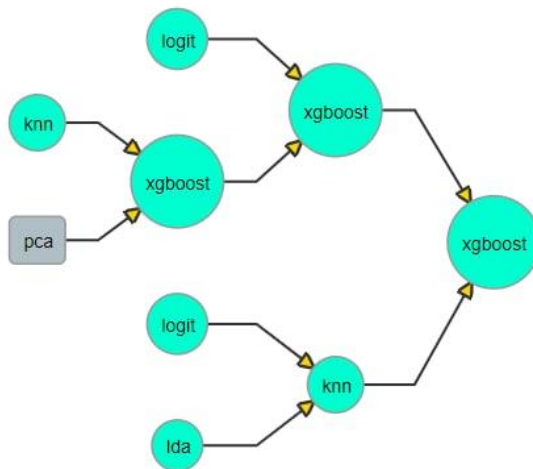
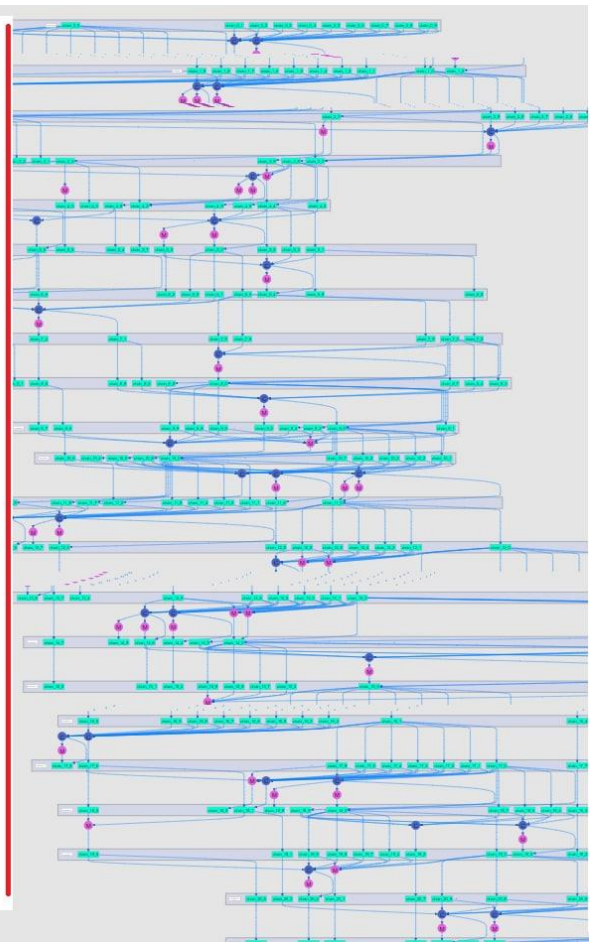


The screenshot shows the GitHub repository page for `nccr-itmo / FEDOT`. The repository has 7 unwatched items, 190 stars, and 24 forks. The main content area displays a file tree for the `master` branch, which has 62 branches and 4 tags. The file tree includes:

- `.github/workflows`: edit requirements (#346), 23 days ago
- `cases`: Prototype of GPU evaluation (#327), yesterday
- `docs`: Prototype of GPU evaluation (#327), yesterday
- `examples`: Prototype of GPU evaluation (#327), yesterday
- `fedot`: Prototype of GPU evaluation (#327), yesterday
- `gpu`: Prototype of GPU evaluation (#327), yesterday
- `test`: Prototype of GPU evaluation (#327), yesterday
- `.codecov.yml`: PyPi release (#180), 8 months ago
- `.gitignore`: Add PyPi publish package trigger in CI (#188), 8 months ago

The right sidebar contains the **About** section, which describes FEDOT as an "Automated modeling and machine learning framework FEDOT" and provides a link to `fedot.readthedocs.io`. It also lists tags such as `machine-learning`, `automation`, `genetic-programming`, `hyperparameter-optimization`, `evolutionary-algorithms`, `multimodality`, `automl`, `automated-machine-learning`, `parameter-tuning`, and `structural-learning`. Below the **About** section is the **Releases** section, which shows the latest release `v0.3.1` as the `Latest` version, released on 30 May, with 3 releases in total.

# Бонус - визуализация эволюции in near future



Интерактивный редактор  
графовых моделей

Интерактивная  
визуализация  
эволюции

## FEDOT

<a href="https://github.com/nccr-itmo/FEDOT">https://github.com/nccr-itmo/FEDOT</a>	Ядро фреймворка, примеры и бенчмарки
<a href="https://www.youtube.com/watch?v=RjbuV6i6de4">https://www.youtube.com/watch?v=RjbuV6i6de4</a>	Intro-видео с описанием фреймворка
<a href="https://t.me/NSS_group">https://t.me/NSS_group</a>	Новости, обновления, релизы
<a href="https://itmo-nss-team.github.io">https://itmo-nss-team.github.io</a>	Natural Systems Simulation Lab



# Спасибо за внимание

mail: [nnikitin@itmo.ru](mailto:nnikitin@itmo.ru)

IT'sMO *re than a*  
UNIVERSITY