

# Label suggestor

Проект выполнили студенты группы М8О-408Б-21

Гришин Алексей

Абдувохидов Эльдар

# Бизнес цель проекта

- **Основной задачей** нашего проекта является автоматическое назначение лейблов для Issues на GitHub. Эта задача направлена на оптимизацию процесса управления задачами в командах, работающими над open source проектами и использующими такие системы управления проектами, как Issues на GitHub
- **Основная цель проекта** - упростить процесс навигации по задачам путем сокращения времени на ручное назначение лейблов.

# Задачи проекта

- Разработать архитектуру, которая будет удовлетворять следующим требованиям:
  - **Масштабируемость:** система должна быть способна к гибкому наращиванию мощностей в случае увеличения нагрузок;
  - **Отказоустойчивость:** при выходе из строя одних компонент, вся система должна оставаться работоспособной;
- Провести обучение модели для классификации тегов на основе информации об Issue
- Подготовить сервисы для дальнейшего деплоя: подготовка docker образов
- Выполнить деплой сервисов на облачной платформе cloud.ru

# Цели МО

## 1. Максимизация точности классификации лейблов

Для улучшения опыта работы с сервисом можно сделать упор на точность классификации. То есть предлагать лейблы, которые наиболее точно соответствуют содержанию Issues

### **Преимущества:**

- Высокая релевантность предложенных лейблов, из чего следует удовлетворение потребностей разработчиков

### **Недостатки:**

- Возможные задержки при обработке из-за использования сложных и трудозатратных подходов

# Цели МО

## 2. Максимизация скорости обработки

С другой стороны, положительный эффект на опыт работы с системой со стороны пользователей может оказать быстрый отклик. То есть, можно сделать упор на то, чтобы быстро предлагать подходящие лейблы даже если из-за этого придется пожертвовать точностью.

### **Преимущества:**

- Ускорение процесса подбора лейблов для Issues

### **Недостатки:**

- Возможное снижение точности

# Цели МО

## 3. Комбинированный подход

Одной из целей может также являться гибридный подход. При таком подходе основной целью является поиск оптимального баланса между точностью и скоростью.

### **Преимущества:**

- Гибкость в настройке системы для различных проектов: где-то время отклика не столь важно по сравнению с точностью предсказания, где-то – наоборот

### **Недостатки:**

- Более сложная реализация и настройка

# Цели МО

## Вывод

Мы выбираем комбинированный подход, так как он позволяет гибко настроить ценность каждой из целей конкретно для каждого проекта и является наиболее универсальным для различных типов проектов.

# Определение входных и выходных данных

Система автоматического назначения лейблов для Issues принимает на вход текстовое описание задачи и метаданные, а на выходе возвращает список предложенных лейблов.





# Выбор категории МО

Для автоматического назначения лейблов к Issues на GitHub наиболее подходящей категорией машинного обучения является задача **многоклассовой классификации**. В данном случае система обучается на размеченных данных, где каждому Issue соответствует один или несколько лейблов, и затем предсказывает наиболее вероятные лейблы для новых Issues.

# Исходные данные

Перед началом работы необходимо понять, какие данные нам доступны. Так как наша система будет помогать подбирать лейблы на этапе создания Issue, то соответственно мы можем использовать только те данные, которые доступны в этот промежуток времени.

По этой причине не рассматриваются такие категории, как:

- **Комментарии к Issues** - комментарии, относящиеся к текущему Issue, оставленные другими пользователями
- **События Issues** - изменения статуса Issue, присвоение меток и другие действия по изменению информации

# Исходные данные

## Данные об Issue

- Текстовое описание:
  - Заголовок - краткое описание Issue
  - Описание - детальное описание Issue в *формате Markdown*
- Пользователь, создающий Issue
- Пользователь, на которого назначена задача
- Уже проставленные лейблы - список меток, уже присвоенных Issue при создании
- Время создания Issue
- Проект - ID проекта, к которому относится Issue
- Milestone - этап разработки, к которому относится Issue

# Исходные данные

## Данные об пользователе

- ID - уникальный идентификатор пользователя на GitHub
- Никнейм - публичное имя пользователя на GitHub

# Исходные данные

## Данные об проекте

- ID - уникальный идентификатор проекта
- Участники - список ID пользователей, которые являются участниками проекта
- Лейблы - набор доступных лейблов для Issue в рамках проекта
- Milestones - набор доступных Milestone-ов в проекте

# Формирование признаков

После сбора данных необходимо провести их очистку и подготовку для использования в модели машинного обучения. В нашем проекте используются следующие шаги обработки:

1. Обработка текстовых данных
2. Обработка текста в формате Markdown
3. Обработка лейблов
4. Обработка несуществующих данных

# Формирование признаков

## Обработка текстовых данных

- Исключение часто встречающихся слов, не несущих значимой информации (например, "и", "в", "на" и т. д.)
- Приведение к нижнему регистру для унификации текста
- Приведение слов к их базовой форме для уменьшения разнообразия форм одного и того же слова
- Преобразование текста в числовой вектор с помощью языковых моделей. Для этой задачи мы выбрали модель BERT.
- Использовать эмбединги текста как признаки.

# Формирование признаков

## Обработка текста в формате Markdown

- Удаление ссылок на медиа
- Удаление стилей (подчеркивание, выделение жирным или курсивом)
- Форматирование для того, чтобы текст имел одинаковую структуру



# Формирование признаков

## Обработка лейблов

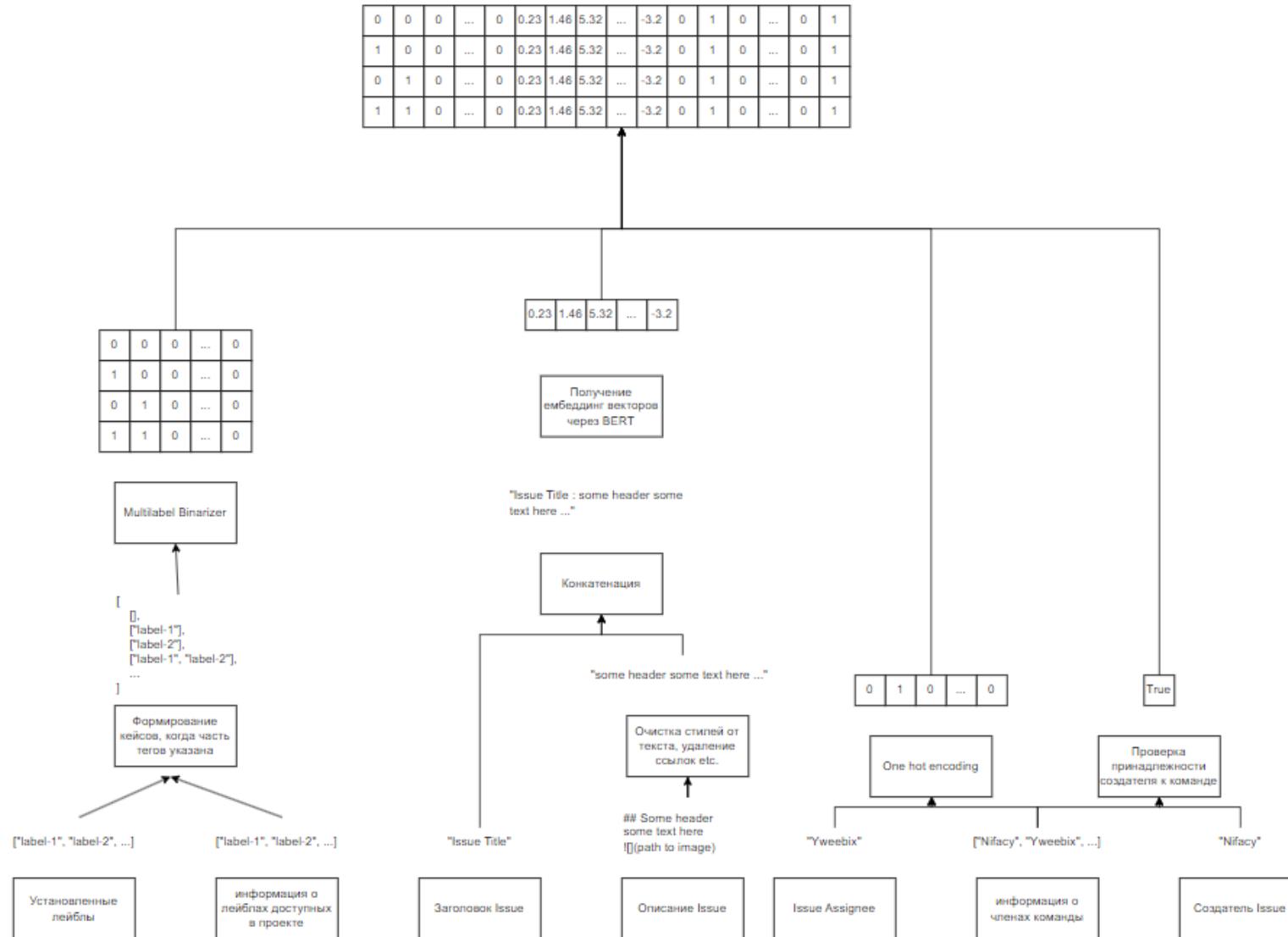
- Некоторые данные, такие как набор лейблов, необходимо преобразовать в числовой формат с использованием методов кодирования. Мы выбрали multilabel encoding.
- Также, для того, чтобы модели было предоставлено больше информации мы сгенерировали набор подмножеств лейблов, симулирующих ситуацию, когда часть тегов уже проставлены.

# Формирование признаков

## Обработка несуществующих данных

- Может возникнуть ситуация, когда в Issue по каким-то причинам будет находиться несуществующие лейблы или они могут быть прикреплены к несуществующему Milestone. В этом случае необходимо эту информацию удалить.

# Формирование признаков



# Важность выбранных признаков

## Принадлежность пользователя команде

Это важно, так как зачастую, issue создаваемые членами команды имеют другое назначение по сравнению с issue, создаваемыми сторонними пользователями. Сторонние пользователи часто используют issue при обнаружении багов или для задания вопросов, в то время как члены команды - для группировки задач.

# Важность выбранных признаков

## Лейблы, отмеченные в Issue

Это важно, так как дает больше информации о назначении issue. Таким образом, итоговый пользователь может даже помочь модели более точно определить итоговый набор лейблов для Issue.

# Важность выбранных признаков

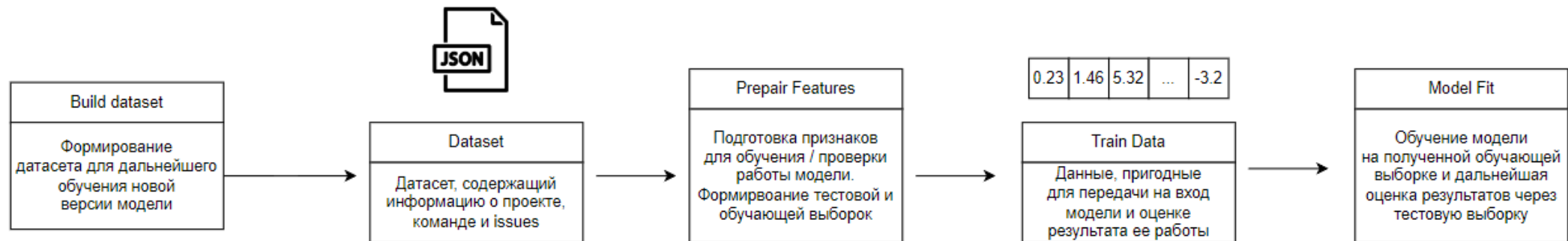
## Пользователь, на которого назначена задача

Это важно, так как зачастую каждый член команды ответственен за некоторую часть проекта, которому могут соответствовать некоторые лейблы. Таким образом, зная, кому назначена задача, мы можем точнее определить, какой тег ему лучше подобрать.

# Обучение модели

Процесс обучения модели в нашем проекте довольно стандартный. Его можно разделить на 3 этапа:

1. На первом этапе мы собираем данные с Github, чтобы в дальнейшем обучить на них новую версию модели
2. На втором этапе мы подготавливаем данные таким образом, чтобы они могли быть переданы на вход модели
3. На последнем этапе происходит непосредственно обучение модели и дальнейшая регистрация новой версии



# Обучение модели

## Почему был использован MLFlow?

Касаемо процесса обучения и дальнейшей работы с итоговой моделью мы решили воспользоваться таким сервисом как MLFlow. Его использование обосновано следующими причинами:

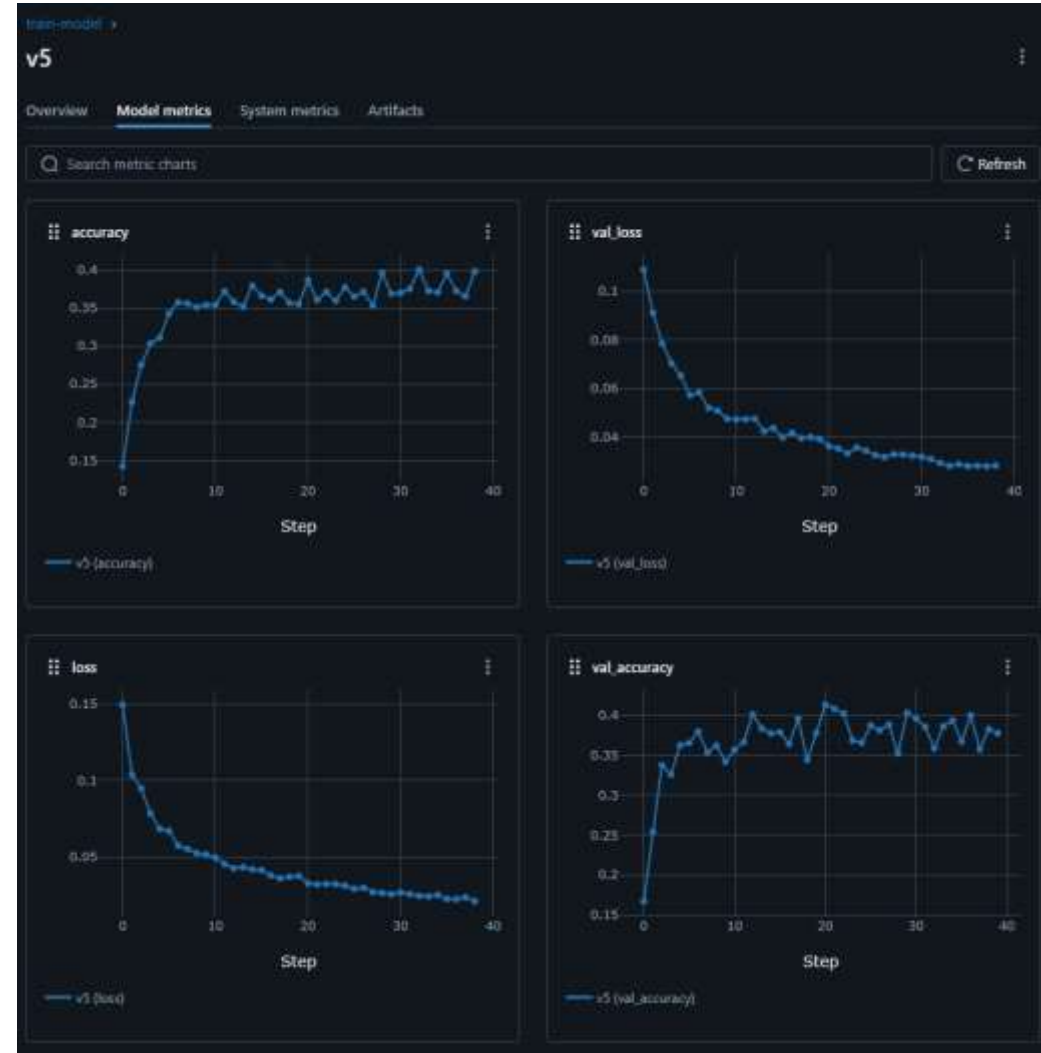
1. **Формализация цикла машинного обучения** – MLFlow помогает формализовать процесс обучения моделей, что делает его более структурированным и понятным;
2. **Артефакты** – MLFlow отлично подходит как система для хранения артефактов, полученных в ходе обучения модели, таких как датасеты, параметры, при которых происходил запуск etc.
3. **Model Registry** – одной из отличительных черт MLFlow является поддержка версионирования моделей, что облегчает и делает процесс использования модели контролируемым



# Обучение модели

## Пример метрик модели

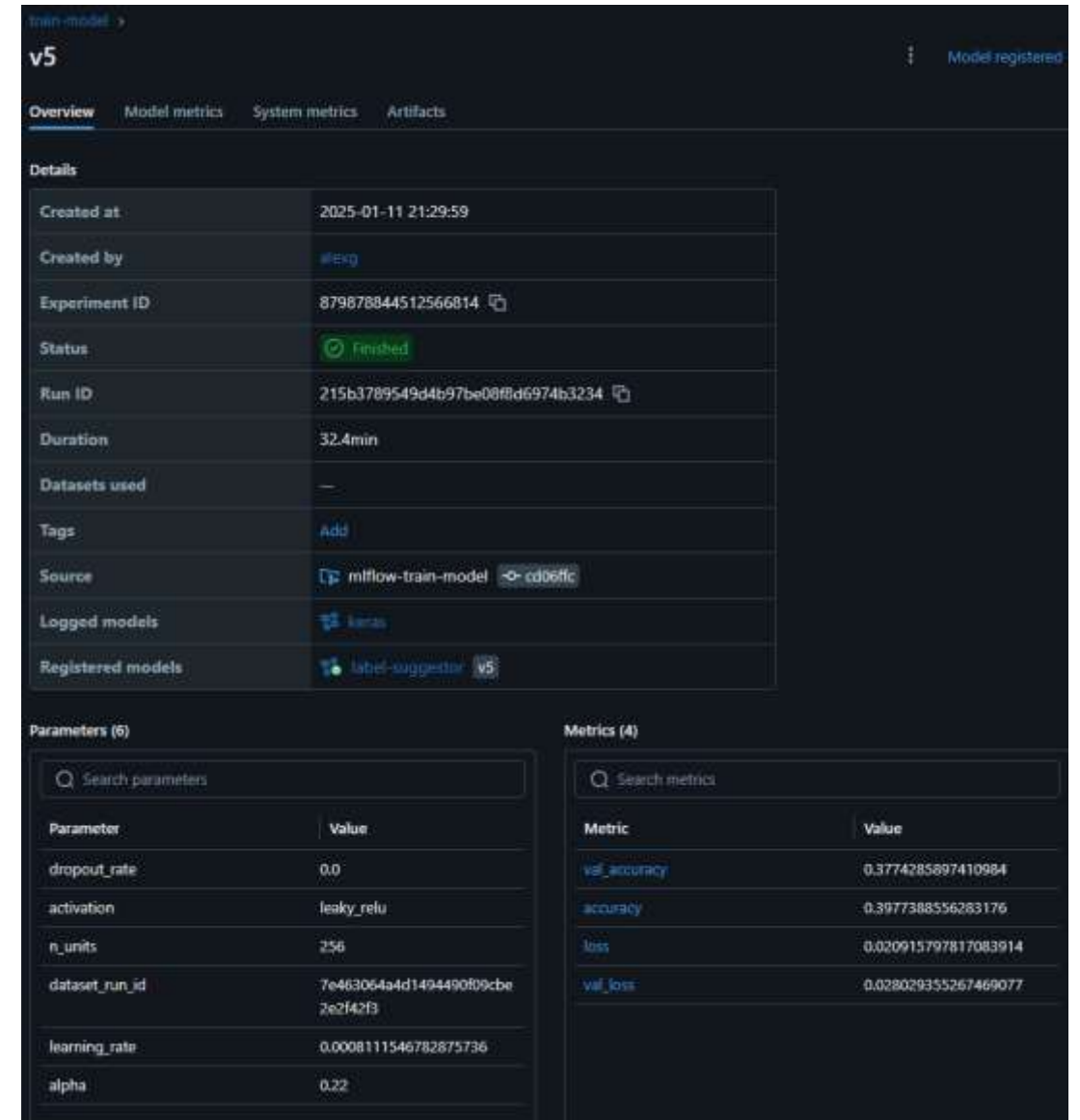
Использование MLFlow позволило с легкостью настроить процесс сбора метрик при обучении модели. Справа предоставлен пример метрик, полученных в ходе обучения одной из версий модели



# Обучение модели

## Пример логирования гиперпараметров

Также использование MLFlow позволило обеспечить логирование гиперпараметров, при которых модель выдавала наилучшие результаты, что позволяет в дальнейшем с легкостью воспроизводить процесс обучения.



The screenshot shows the MLFlow training run interface for a model named 'v5'. The interface is divided into several sections: Overview, Model metrics, System metrics, and Artifacts. The Overview section is currently selected and displays the following details:

Parameter	Value
Created at	2025-01-11 21:29:59
Created by	#Esg
Experiment ID	879878844512566814
Status	Finished
Run ID	215b3789549d4b97be08f0d6974b3234
Duration	32.4min
Datasets used	—
Tags	Add
Source	mlflow-train-model cd06ffc
Logged models	keras
Registered models	label-suggestor v5

Below the details section, there are two panels: Parameters (6) and Metrics (4).

**Parameters (6)**

Parameter	Value
dropout_rate	0.0
activation	leaky_relu
n_units	256
dataset_run_id	7e463064a4d1494490f09cbe2e2f42f3
learning_rate	0.0008111546782875736
alpha	0.22

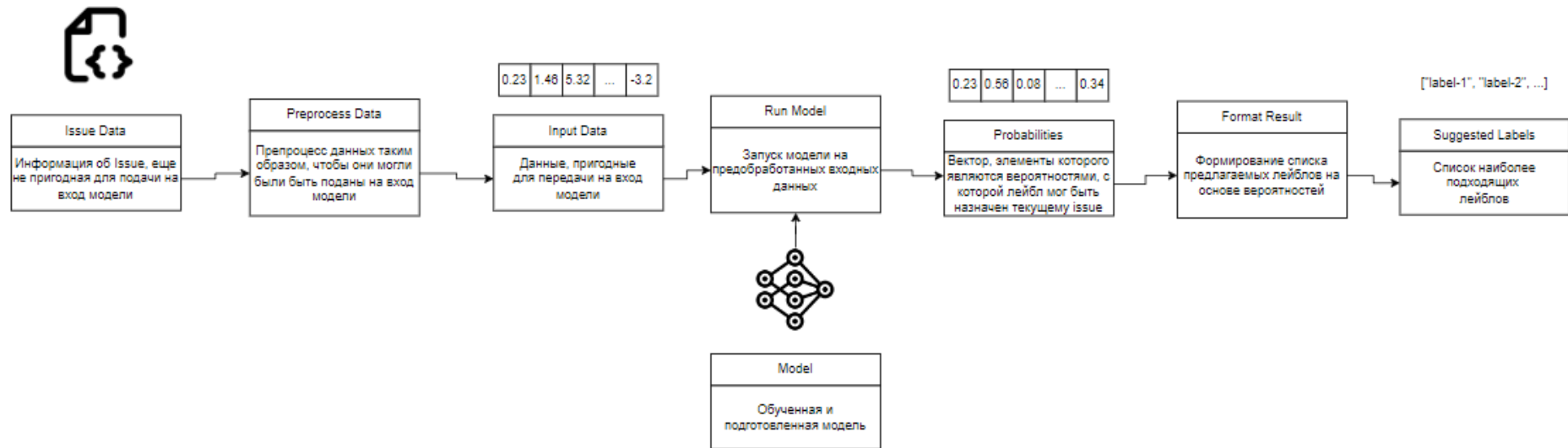
**Metrics (4)**

Metric	Value
val_accuracy	0.3774285897410984
accuracy	0.3977388556263176
loss	0.020915797817083914
val_loss	0.028029355267469077

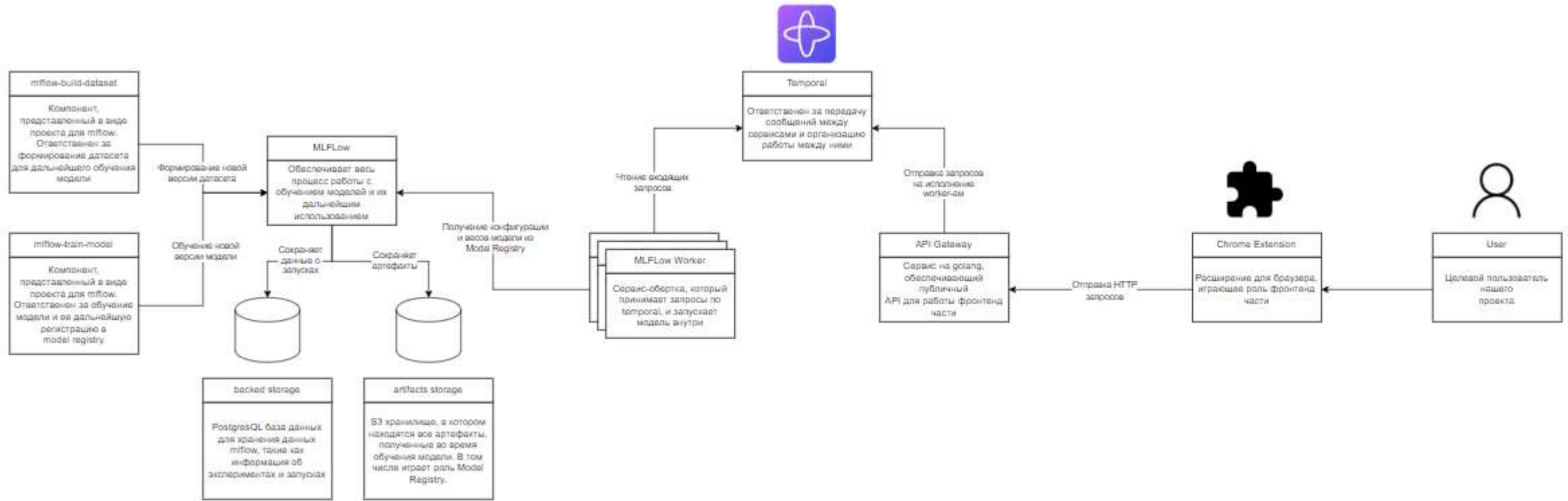
# Как работает модель?

Сама модель имеет довольно простую форму. Ее работу можно разделить на 3 этапа:

- **Подготовка данных:** данный этап ответственен за то, чтобы сформировать признаки из входных данных. В последующем эти признаки будут отправлены следующему компоненту.
- **Классификация лейбла:** на этом этапе подготовленные признаки отправляются на вход модели-классификатору. В качестве результата мы получаем вектор, содержащий вероятности, с которой мог бы быть поставлен каждый из лейблов к Issue.
- **Формирование результата:** последним этапом является формирование списка предлагаемых лейблов на основе их вероятностей, полученных с прошлого этапа.



# Архитектура проекта



# Архитектура проекта

## Worker Service

Данный сервис ответственен за обработку клиентских запросов. Он также является сервисом-оберткой для обученной модели. Сам Worker работает по протоколу RPC с использованием фреймворка gRPC. Входные данные сервис получает по очереди через Temporal. Аналогично, используя очередь в Temporal, сервис возвращает результат обработки запроса. При инициализации, сервис обращается в MLFlow сервер, чтобы скачать версию модели и в дальнейшем использовать ее для обработки запросов.

### Процесс работы:

1. При инициализации worker обращается в MLFlow и скачивает модель
2. Worker получает запрос через Temporal
3. Worker выполняет препроцессинг входных данных
4. Worker отправляет данные в модель и выполняет формирование результата
5. Worker отправляет результат через Temporal

# Архитектура проекта

## API Gateway

Данный сервис ответственен за предоставление API бэкенд части. Его работа заключается в получении HTTP запросов от клиентской части и дальнейшему формированию запроса в Worker.

### Процесс работы:

1. получает запрос от клиентской части
2. формирует запрос и отправляет его в очередь в Temporal
3. Получает ответ от worker-а через Temporal
4. Возвращает результат в ответ на запрос от клиентской части

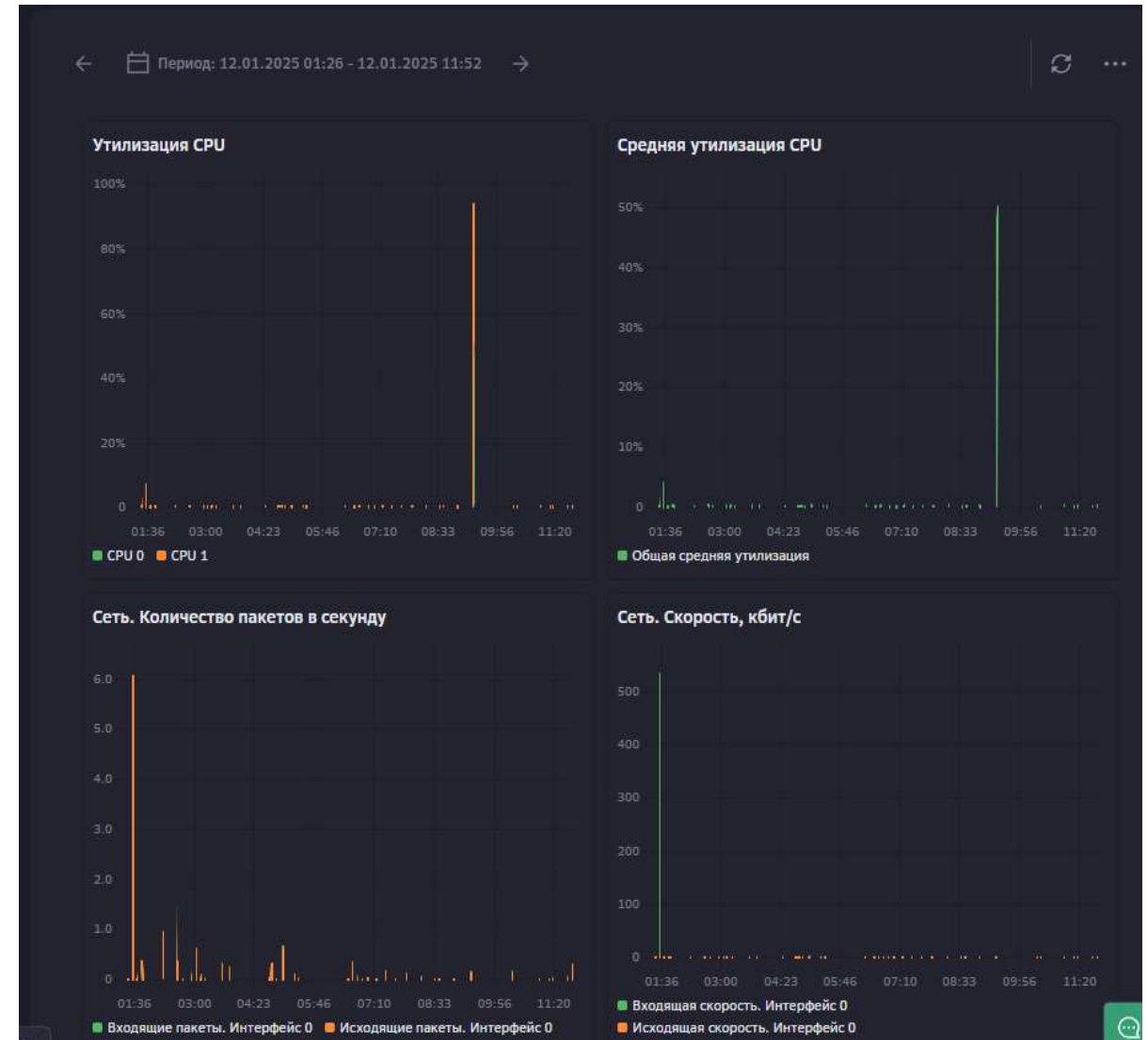
# Почему temporal?

Для общения сервисов мы использовали такой сервис как Temporal. Его выбор обусловлен по следующим причинам:

1. **Удобство:** Temporal очень удобен в контексте настройки и его дальнейшего деплоя. Он отлично подходит как для локального использования, так и для использования в production;
2. **Богатый функционал:** temporal предоставляет богатый функционал, что делает работу с ним более полной. Например, temporal предоставляет UI сервер для визуального отображения состояния. Наличие SDK для большого количества языков также оказывает положительное влияние, особенно при разработке сервисов, написанных на разных языках программирования;
3. **Управление состоянием и долговечность:** temporal автоматически сохраняет состояние выполнения задач, обеспечивая тем самым их надежное завершение даже в случае сбоев.

# Мониторинг и поддержка системы

Также, после деплоя системы необходимо поддерживать и контролировать ее состояние, а также вовремя реагировать на случающиеся сбои. С этой целью важно вести мониторинг ключевых метрик, таких как нагрузка CPU, сети и т. д. Справа предоставлен пример технических мониторингов.





# Решение проблем при эксплуатации

1. **Высокая нагрузка:** Одной из проблем, которая может возникнуть при эксплуатации системы может быть повышенная нагрузка. Данная проблема решается через горизонтальное масштабирование, путем создания новых инстансов сервиса worker.
2. **Деградация модели:** Со временем модель может выдавать некорректные результаты, так как обучалась она на устаревших данных. Данная проблема решается через MLFlow путем обучения новой версии модели на обновленном датасете, ее тестировании и дальнейшем использовании в production.

# Пример использования



[Демонстрация](#)



Add a title

Title

Add a description

Write

Preview

H

B

I

≡

<>

🔗

≡

≡

🔗

📎


@

🗨️

...

is this a bug

 Markdown is supported

 Paste, drop, or click to add files

Submit new issue

Assignees

No one assigned

Labels

[area/artifacts](#)

[area/examples](#)

[bug](#)

[language/r](#)

None yet

Projects

None yet

Milestone

No milestone

Development

Shows branches and pull requests linked to this issue.

 Remember, contributions to this repository should follow its [contributing guidelines](#) and [security policy](#).