СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

AutoML – автоматическое машинное обучение.

NLP – обработка естественного языка.

NLU – понимание естественного языка.

Эмбеддинг – .

Промпт – .

Инференс

Введение

В последние годы наблюдается бурный рост интереса к диалоговым системам на основе искусственного интеллекта (чат-ботам, голосовым помощникам и так далее). Так, по данным трендов, интерес к голосовым технологиям AI вырос почти втрое за пять лет[1]. Диалоговые системы внедряются в бизнес-процессы и повседневную жизнь, однако создание их интеллектуальной части – модели определения намерения пользователя (intent classification) – остается сложной задачей. Такой модуль является ключевым компонентом системы, позволяя автоматически выявлять цель запроса пользователя, но учет специфики различных доменов серьезно затрудняет разработку универсальной модели. Проектирование и тонкая настройка модели интентов требуют значительных экспертных усилий в области NLP и ML. Поэтому актуальной представляется автоматизация данного процесса – создание универсальных решений, способных уменьшить долю ручной работы и упростить разработку моделей классификации интентов. Автоматизированные подходы к машинному обучению (AutoML) обещают значительно сократить объем ручного труда за счет автоматического подбора оптимальных моделей и параметров, что особенно важно для быстро растущей области диалоговых систем.

На сегодняшний день для задачи классификации интентов накоплен внушительный арсенал методов. Традиционно применяются алгоритмы классического машинного обучения, такие как наивный tf-idf[1], а также подходы на основе k-ближайших соседей и ансамблевые методы (например, градиентный бустинг). С развитием глубокого обучения все более широко используются нейросетевые модели, таких как BERT[2], которые достигают высоких показателей качества на задачах. Параллельно развиваются технологии AutoML, автоматизирующие выбор моделей и настройку гиперпараметров. Тем не менее, несмотря на прогресс отдельных компонентов, целостных универсальных AutoML-фреймворков, специально ориентированных на определение интентов пользователя, предложено немного. Существующие решения зачастую требуют участия эксперта для каждой

[1] https://www.verloop.io/blog/100-best-chatbot-statistics

новой предметной области, что указывает на необходимость разработать более обобщенный подход.

В связи с этим актуальной является проблема отсутствия универсального, масштабируемого и эффективного AutoML-решения для классификации интентов, способного автоматически адаптироваться к разным доменам без глубокого участия человека-эксперта.

Цель исследования заключается в разработке такого универсального AutoML-фреймворка, который способен автоматически подбирать оптимальные модели и их конфигурации для классификации интентов пользователя. Разработанное решение будет протестировано на различных корпусах данных (наборы пользовательских запросов), а его эффективность сопоставлена с результатами ручной настройки моделей, чтобы оценить выигрыш от автоматизации.

Для достижения поставленной цели в работе решены следующие задачи:

1. Провести обзор существующих фреймворков и библиотек, применяемых для построения моделей машинного обучения, включая решения для задач классификации пользовательских намерений;
2. Выполнить анализ современных алгоритмов и подходов к задаче определения пользовательских намерений, включая традиционные методы машинного обучения и нейросетевые архитектуры;
3. Разработать концепцию и архитектуру собственного фреймворка создания моделей машинного обучения для классификации пользовательских намерений;
4. Реализовать программную часть фреймворка с возможностью автоматизированной настройки моделей и выбора признаков;
5. Провести экспериментальное исследование эффективности фреймворка на нескольких датасетах из разных предметных областей;
6. Сравнить результаты, полученные с использованием разработанного фреймворка, с качеством моделей, настроенных вручную, и провести анализ полученных результатов.

Практическая значимость работы состоит в том, что созданный AutoML-фреймворк может быть непосредственно применен при разработке реальных диалоговых систем – чат-ботов, голосовых ассистентов, систем

клиентской поддержки – и других NLP-приложений. Использование такого инструмента позволит ускорить внедрение новых сервисов и снизить порог вхождения для разработчиков за счет автоматизации подбора оптимальной модели под конкретный набор интентов.

Научная новизна исследования определяется интеграцией современных методов автоматизированного машинного обучения в единой специализированной архитектуре, ориентированной на задачу классификации интентов. В предлагаемом решении объединяются передовые подходы, включая трансформерные модели и методы обучения с малым количеством примеров, в рамках одного AutoML-фреймворка. Такое сочетание технологий нацелено на достижение высокой точности и устойчивости модели при минимальном ручном вмешательстве, что ранее не было реализовано в полной мере для задачи определения интентов пользователя.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Определение намерений пользователя

Классификация намерений – это задача сопоставления высказывания пользователя с предопределенной меткой намерения (семантической категорией цели пользователя). Например, запрос “Какая погода будет завтра?” может быть классифицирован как запрос погоды. Эта способность является ключевым компонентом понимания естественного языка (NLU) в диалоговых системах, позволяя чат-ботам, виртуальным помощникам и другим агентам искусственного интеллекта понимать, чего хочет пользователь, и соответствующим образом реагировать. Классификация намерений уходит корнями в ранние разговорные диалоговые системы (например, телефонное обслуживание клиентов) и с тех пор получила повсеместное распространение в самых разных областях - от личных помощников и ботов поддержки клиентов до систем медицинских и юридических консультаций.

Ранние методы были основаны на правилах, которые разрабатывались вручную, или на классическом машинном обучении с добавлением дополнительных функций. Однако с развитием области преобладать стали статистические методы, которые основываются на анализе данных. Сначала они использовали традиционные алгоритмы машинного обучения, а затем — методы глубокого обучения. Также мы наблюдаем расширение сферы применения: от простой классификации с закрытым набором параметров, когда каждый запрос должен относиться к одному из известных намерений, до более сложных сценариев. Например, к многоцелевой классификации, обнаружению намерений с открытым доменом или открытым набором параметров (когда запрос не соответствует ни одному из известных намерений), а также к распознаванию намерений с минимальным количеством попыток или вообще без них с помощью мощных генеративных моделей. Классификация намерений – это задача сопоставления высказывания пользователя с предопределённой меткой, или интентом, отражающим семантическую цель запроса. Например, запрос “Какая погода будет завтра?” может быть классифицирован как запрос погоды. Эта способность

является ключевым компонентом понимания естественного языка (NLU) в диалоговых системах, позволяя чат-ботам, виртуальным помощникам и другим агентам искусственного интеллекта понимать, чего хочет пользователь, и соответствующим образом реагировать. Классификация намерений уходит корнями в ранние разговорные диалоговые системы (например, телефонное обслуживание клиентов) и с тех пор получила повсеместное распространение в самых разных областях - от личных помощников и ботов поддержки клиентов до систем медицинских и юридических консультаций.

Изначально системы классификации намерений строились на вручную заданных правилах и классических алгоритмах машинного обучения с набором признаков. С развитием технологий появились статистические методы и глубокие нейронные сети. Постепенно задачи усложнились – появились мульти-интентная классификация и открытые домены.

Также появилась проблема с запросами, которые не соответствуют ни одному из известных интентов (Out-of-Scope (OOS)) и требуют специальной обработки, чтобы избежать некорректных ответов.

Типичные примеры OOS-запросов: «Какой сейчас курс евро?» в погодном ассистенте, «Как оформить ипотеку?» в кино-ассистенте, «Расскажи, что я ел на прошлой неделе» в системе, не хранящей историю питания. При таких запросах система вежливо отказывается («Извините, я не могу помочь с этим запросом«), предлагает альтернативы или перенаправляет в службу поддержки, а сами OOS-записи сохраняются для расширения набора интентов в будущем.

1.2 Методы автоматического машинного обучения

Автоматизированное машинное обучение (AutoML) относится к автоматизации полного процесса применения методов машинного обучения для решения реальных задач. Вместо того чтобы вручную выбирать алгоритмы, настраивать гиперпараметры, разрабатывать архитектуры моделей и создавать признаки, система AutoML автоматически принимает эти решения на основе данных. Мотивация для развития AutoML вытекает из бурного роста применения машинного обучения и стремления «демократизировать» машинное обучение – сделать современные техники доступными даже для неспециалистов. Модели машинного обучения зача-

стую чувствительны к множеству параметров (тип модели, архитектура, настройки гиперпараметров, предварительная обработка признаков и так далее), и нахождение оптимальной конфигурации часто требует кропотливого перебора даже для экспертов. Эта проблема особенно заметна в глубоком обучении, где выбор правильной архитектуры сети и стратегии обучения может определять конечное качество модели. Цель AutoML – автоматизировать принятие этих решений, позволяя пользователю просто предоставить данные, а система подбирает оптимальную модель. Данный обзор литературы предоставляет академический анализ AutoML с основным упором на его применение в обработке естественного языка (NLP), а также включает как фундаментальные работы, так и последние разработки. Мы рассмотрим историческую эволюцию и мотивации AutoML, ключевые технические компоненты, ведущие фреймворки и системы, особенности применения AutoML в задачах NLP (например, классификация текстов, маркировка последовательностей, языковое моделирование), сравнительный анализ производительности и существующие бенчмарки, а также новые тенденции и направления исследований (например, интеграция с фундаментальными моделями, обучение с малым количеством примеров, объяснимость моделей). Обзор ссылается на рецензируемые публикации и академические источники.

1.2.1 H2O

H2O[3] – является платформой машинного обучения с открытым исходным кодом, разработанной для автоматизации процесса контролируемого обучения. Она ориентирована на выполнение таких задач, как бинарная классификация, многоклассовая классификация и регрессия. Основная функция AutoML в H2O заключается в обучении широкого спектра алгоритмов, включая градиентные повышающие машины, случайные леса, глубокие нейронные сети и обобщенные линейные модели. Затем эти модели объединяются в ансамбль для получения наилучших предсказаний. Результатом работы AutoML является таблица лидеров — ранжированный список моделей по показателям производительности, из которого можно выбрать оптимальную модель для развертывания.

Процесс автоматизации в H2O ограничен по времени, что позволяет пользователю задать максимальное время выполнения или количество

моделей. Система обучает столько моделей, сколько возможно в рамках установленных ограничений. В отличие от более сложных методов оптимизации гиперпараметров (например, байесовской оптимизации), H2O использует случайный перебор моделей, полагаясь на разнообразие моделей и ансамблевую технику для достижения высокой производительности. Особенностью платформы является автоматическое создание двух сложенных ансамблей: один включает все обученные модели, другой — только лучшие модели каждого семейства алгоритмов. Этот подход позволяет повысить точность предсказаний без ручной настройки.

H2O эффективно обрабатывает большие массивы данных за счет распределения вычислений по нескольким ядрам или узлам кластера. Платформа реализована на языке Java и предоставляет API для Python, R и других языков, что позволяет интегрировать её в различные среды. Результаты работы легко интерпретируемы: на выходе формируется ранжированный список моделей с указанием показателей производительности и времени обучения. Кроме того, встроенные инструменты объяснения моделей позволяют пользователям получать такие пояснения, как важность переменных, графики частичной зависимости и значения SHAP для лучших моделей. Таким образом, H2O обеспечивает возможность построения множества моделей за короткий промежуток времени, что особенно актуально при работе с большими объемами данных.

1.2.2 LightAutoML

LightAutoML[4] (LAMA) — это облегчённый фреймворк AutoML с открытым исходным кодом, предназначенный для моделирования табличных данных. Основное назначение LAMA — автоматическая генерация конвейеров для структурированных данных с акцентом на скорость и эффективность использования вычислительных ресурсов. Он поддерживает задачи бинарной и многоклассовой классификации, а также регрессию. Изначально ориентированный на работу с табличными данными, LightAutoML расширил свои возможности и теперь поддерживает текстовые признаки.

Фреймворк автоматически выполняет предварительную обработку данных, включая очистку и кодирование отсутствующих значений, вывод типов признаков и их отбор в рамках конвейера. Гиперпараметры моделей настраиваются автоматически. LightAutoML предоставляет готовые пресе-

ты конвейеров (например, «TabularAutoML»), которые обеспечивают быстрое развертывание моделей с минимальным вмешательством пользователя. Также доступны гибкие настройки для создания собственных конвейеров с учетом специфики задачи.

Отличительной чертой LightAutoML является параллельное обучение нескольких конвейеров, результаты которых объединяются с использованием ансамблевых методов. Это может быть простое усреднение или более сложное построение, при котором модели одного уровня используют предсказания предыдущего уровня в качестве входных данных. Также важной концепцией является разделение этапов чтения и предварительной обработки: компонент «Reader» проверяет исходный набор данных и определяет необходимые преобразования для различных типов признаков. Это гарантирует надежность и автоматизацию оценки модели.

1.2.3 AutoGluon

AutoGluon[5] – комплексный инструментарий AutoML с открытым исходным кодом, который поддерживает широкий спектр задач машинного обучения, включая прогнозирование табличных данных, компьютерное зрение, обработку естественного языка и прогнозирование временных рядов. Фреймворк предоставляет специализированные API для каждой задачи, например, TabularPredictor и TextPredictor, что упрощает использование в различных областях.

AutoGluon поддерживает обучение различных моделей: от древовидных алгоритмов (LightGBM, XGBoost[6], CatBoost[7]) до нейронных сетей (например, трансформеров для текста и сверточных сетей для изображений), а также простых моделей, таких как k-ближайших соседей и линейные модели. Пользователь может настроить гиперпараметры, выбрать конкретные модели для обучения и использовать предустановленные конфигурации. Таким образом, AutoGluon предоставляет гибкие возможности для настройки обучения с минимальным объемом кода.

1.2.4 FEDOT

FEDOT[8; 9](Flexible Evolutionary Design of Optimal Trees) – фреймворк AutoML с акцентом на оптимизацию конвейеров с помощью эволюционных алгоритмов. Разработанный лабораторией моделирования природ-

ных систем Университета ИТМО, он предназначен для автоматизации полного жизненного цикла машинного обучения: от предварительной обработки данных до построения и оптимизации моделей.

Основная идея FEDOT – создание составных конвейеров с помощью генетических алгоритмов. Конвейер представлен в виде направленного ациклического графа (DAG), узлы которого могут быть как преобразованиями данных, так и моделями. Эволюционный оптимизатор, известный как «GOLEM», генерирует начальную популяцию случайных конвейеров и затем улучшает их с помощью мутаций и скрещивания. В результате создаются оптимальные конвейеры, адаптированные к конкретной задаче.

FEDOT поддерживает работу с различными типами данных (табличные, текстовые, графовые) и обеспечивает гибкость настройки. Инструменты анализа позволяют исследовать чувствительность компонентов конвейера, а также оценивать влияние отдельных моделей на итоговую производительность. Фреймворк поддерживает экспорт оптимальных конвейеров в формате JSON и позволяет интеграцию в производственные среды.

1.2.5 Сравнение алгоритмов

Классификация намерений пользователя представляет собой важную задачу в области обработки естественного языка, требующую использования современных методов машинного обучения и автоматизированных инструментов для построения эффективных моделей. Для реализации данной задачи важно учитывать несколько ключевых критериев: способы обработки текста, поддержка работы с малым набором данных, поддержка выявления намерений вне области определения (Out-Of-Scope, OOS), гибкость настройки параметров, поддержка логирования и возможность использования промптов для энкодеров.

Первым важным критерием является обработка текста, поскольку текстовые данные являются основным источником информации при классификации намерений. Современные модели, такие как трансформеры, демонстрируют высокую точность в задачах NLP благодаря обучению на больших объемах текстов. Поэтому наличие встроенной поддержки текстовых признаков, включая возможность применения эмбеддингов и трансформерных архитектур, является важным аспектом при создании фреймворка.

Вторым значимым критерием является поддержка работы с малым набором данных. В прикладных задачах, связанных с классификацией намерений, часто возникает ситуация, когда количество размеченных данных ограничено. Это особенно актуально при адаптации моделей к новым доменам или редким языковым конструкциям. Поэтому важной характеристикой фреймворка является его способность эффективно работать с малыми наборами данных, например, за счет использования регуляризации или предварительно обученных эмбеддингов.

Не менее важной является поддержка Out-Of-Scope (OOS) – задачи, заключающейся в выявлении запросов пользователя, выходящих за рамки известных классов намерений. Выявление OOS-классов критично для обеспечения надежности и безопасности систем, поскольку позволяет корректно обрабатывать неизвестные или неподдерживаемые запросы. Фреймворки, реализующие данную функциональность, позволяют обучать модели, способные распознавать не только заданные классы, но и детектировать аномальные данные.

Следующим важным аспектом является изменение параметров запуска. В зависимости от задачи, объема данных и доступных вычислительных ресурсов, может потребоваться гибкая настройка процесса обучения. Это особенно актуально при разработке моделей для различных доменов или на основе разнородных данных. Возможность адаптировать параметры позволяет оптимизировать модель как по точности, так и по времени выполнения.

Поддержка логирования является важным компонентом автоматизации машинного обучения, поскольку позволяет отслеживать процесс обучения, хранить промежуточные результаты и проводить анализ моделей. В контексте классификации намерений важно иметь возможность анализировать ошибки и проверять гипотезы о моделях на каждом этапе обучения. Логирование помогает выявлять причины ухудшения качества моделей и отслеживать процессы настройки гиперпараметров, что критично для обеспечения повторяемости экспериментов и объяснимости конечных результатов.

Последним критерием является поддержка промптов для энкодеров, что особенно важно при использовании моделей на основе трансформеров.

В последнее появляются модели, которые поддерживают промпты в зависимости от задачи, которые улучшают качество ее работы. Например, модель e5[10] использует `query:` и `passage:` для создания разных частей эмбеддинга для поиска похожих текста.

Таблица сравнения фреймворков по заданным критериям 1.1.

Таблица 1.1 — Сравнение AutoML фреймворков

Критерий	H2O	LightAutoML	AutoGluon	FEDOT
Способы обработки текста	Нет поддежки из коробки	TF-iDF[1] и эмбеддинг	Эмбеддинг	TF-iDF, эмбеддинг
Поддержка работы с малым набором данных	Не оптимизирован для малых данных	Имеет режимы, позволяющие работать с небольшими наборами данных	Нет поддержки	Может адаптироваться к малым данным
Изменение параметров запуска	Гибкая настройка через API	Настройка через пресеты и конфигурацию, плохо документировано	Можно передавать свой конфиг, плохо документировано	Ограненная настройка
Поддержка логирования во внешние системы	Логирование результатов через интеграцию с H2O Flow	Нет поддержки	Нет поддержки	Нет поддержки
Поддержка промптов для энкодеров	Нет поддержки	Нет поддержки	Нет поддержки	Нет поддежки
Поддержка OOS (out of scope)	Нет встроенной поддержки	Нет встроенной поддержки	Нет поддержки	Нет поддержки

1.3 Нейросетевые методы представления текста

1.3.1 BERT

BERT[2] (Bidirectional Encoder Representations from Transformers) — это языковая модель на основе архитектуры трансформера[11], которая предобучается на задаче маскированного языкового моделирования и предсказания следующего предложения. В отличие от односторонних моделей вроде GPT[12] или неглубоких двунаправленных конкатенаций, таких как ELMo[13], BERT одновременно учитывает и левый, и правый контекст на всех слоях, что обеспечивает более глубокое понимание языка.

В своей базовой конфигурации модель содержит 12 «базовых блоков» (слоёв) трансформера, а размер скрытых представлений в каждом из них равен 768. Входной текст разбивается на токены с помощью WordPiece (словарь из 30 000 токенов), затем в начало последовательности добавляется специальный маркер `[CLS]`, а при подаче пары предложений между ними вставляется `[SEP]`. К каждому токену добавляются позиционные эмбеддинги и эмбеддинги сегментов, указывающие, к какому из предложений он относится. Представление токена `[CLS]` служит свёрнутым вектором для задач классификации, а остальные эмбеддинги используются для задачи span-prediction.

Во время предобучения первая задача — маскирование токенов. 15 % токенов в каждом примере случайно выбирается для маскировки: 80 % из них заменяются на `[MASK]`, 10 % — на случайный токен, и 10 % остаются без изменений. Модель пытается угадать исходные токены, опираясь на полный контекст. Такая схема способствует более устойчивому обучению по сравнению с традиционными слева-направо или справа-налево моделями.

Вторая задача — предсказание следующего предложения: с вероятностью 50 % подаётся пара из действительно идущих друг за другом предложений, а с вероятностью 50 % — два случайных предложения из корпуса. Модель обучается определять, являются ли они смежными, что развивает понимание связности и логики текста (см. рис. 1.3).

Для решения downstream-задач BERT требует лишь добавления небольшой выходной головы: для классификации на токен `[CLS]`, для span-prediction — двух векторов начала и конца и т. д. Затем все параметры

модели дообучаются одновременно, что делает адаптацию универсальной и простой. Абляционные эксперименты показывают, что и двунаправленность внимания, и задача предсказания следующего предложения критически важны: при их исключении эффективность существенно падает, а увеличение глубины и ширины модели даёт стабильный прирост в переносимости представлений.

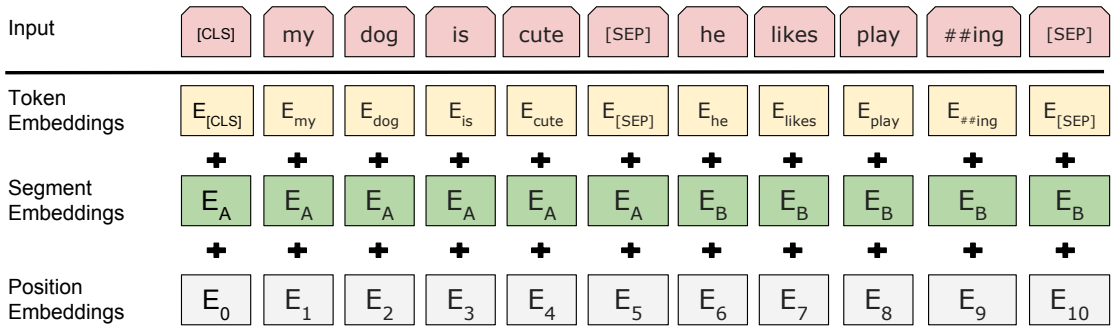

Рисунок 1.1 — Пример преобразования входного текста в эмбеддинги

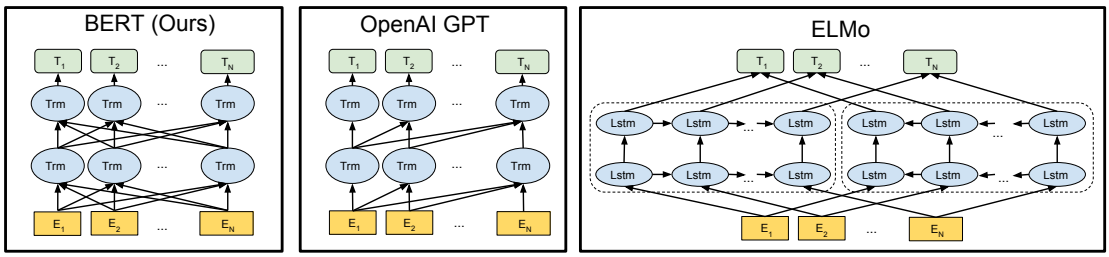

Рисунок 1.2 — Сравнение направленностей внимания ELMo, GPT и BERT

1.3.2 Sentence Transformers

Sentence BERT[14] (SBERT) – модификаця исходной моделей BERT, нацеленная на эффективное построение векторных представлений предложений. В данной работе BERT выступает в роли общего кодировщика, параметры которого разделяются между двумя (или тремя, в случае триплетной версии) ветвями сети, обрабатывающими по отдельности входные предложения. Такое «сиамское» строение (biencoder) позволяет получать фиксированные векторы предложений, сохраняющие богатую семантическую информацию, без необходимости совместной обработки пар предложений на этапе инференса.

Основной этап обучения SBERT заключается в тонкой подгонке предобученного трансформера на разметках задач распознавания естественного вывода (SNLI, Multi-Genre NLI) или семантического сходства (STS).

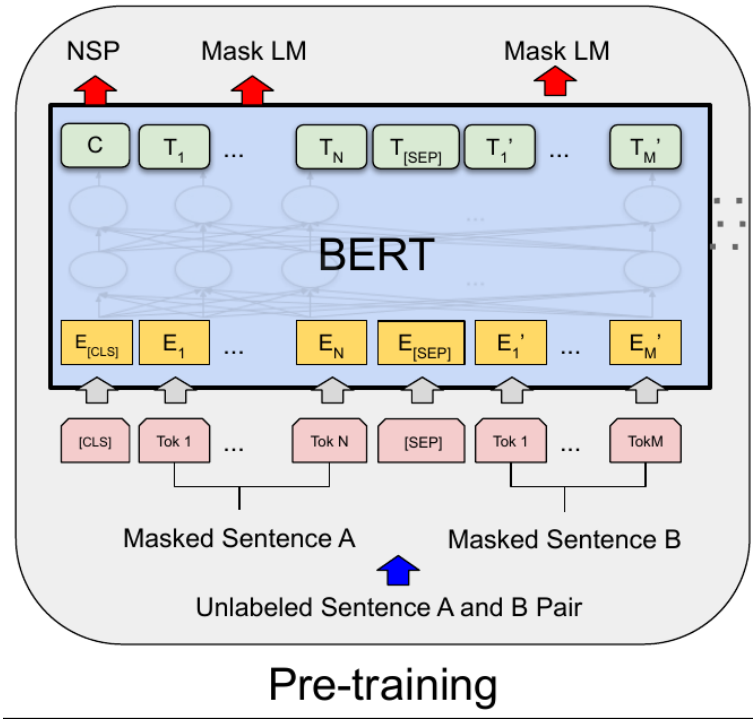

Рисунок 1.3 — Схема задачи предсказания следующего предложения в BERT

После прохождения каждого предложения через общий энкодер применяются операции агрегирования (mean-, CLS- или max-пулинг), формирующие итоговый эмбеддинг. Для оптимизации используются три различных критерия: классификационный (с дополнительным полносвязным слоем и softmax), регрессионный (минимизация MSE на косинусном сходстве) и триплетная функция потерь (гарантирующая, что «анкоры» ближе к «позитивам», чем к «негативам» на заданный порог).

В результате декомпозиции процедуры сравнения пар предложений и предварительного вычисления эмбеддингов SBERT демонстрирует существенное ускорение: поиск ближайших соседей в корпусе из 10 000 предложений, требовавший ранее десятков часов работы перекрёстного энкодера BERT на GPU, сводится к нескольким секундам при использовании SBERT и быстрых алгоритмов косинусного поиска. Это позволяет применять семантический поиск, кластеризацию и извлечение информации в режиме реального времени и на больших масштабах.

Стоит различать две парадигмы работы с парными входами в трансформерах. Cross-encoder принимает на вход конкатенацию двух предложений, обрабатывает их совместно и выдает прямую оценку сходства (или

класс) через полносвязный классификатор — такая схема обеспечивает высочайшую точность, но накладывает квадратичную по размеру корпуса сложность инференса. Biencoder (сиамская или двухветвная модель) кодирует каждое предложение независимо в единое пространство эмбеддингов, после чего сходство вычисляется быстро «на лету» с помощью косинусной меры; это даёт компромисс между качеством и производительностью и лежит в основе SBERT.

1.4 Методы адаптации моделей

1.4.1 P-Tuning

P-Tuning[15] дополняет дискретные подсказки обучаемыми непрерывными эмбеддингами, превращая их в гибридную схему, где модель может автоматически адаптировать ввод под специфику задачи. Вместо жёстко заданных шаблонов к входному тексту добавляется последовательность параметризованных векторов подсказок, которые проходят через облегчённый энкодер (LSTM, MLP или identity) и оптимизируются вместе с моделью или независимо от неё.

Метод решает проблему высокой нестабильности ручных дискретных подсказок, когда даже незначительные изменения формулировки приводят к резкому падению качества. Благодаря обучаемым эмбеддингам P-Tuning снижает разброс результатов при различных вариантах подсказок и позволяет получать более предсказуемый отклик модели.

P-Tuning демонстрирует значительный рост точности и устойчивости на широком спектре задач: от фактического знания (LAMA) до комплексных NLU-бенчмарков (SuperGLUE) и сценариев с ограниченным числом примеров. Этот подход обеспечивает более быструю и надёжную адаптацию моделей к новым задачам без затрат на полный перебор шаблонов.

1.4.2 LoRA

LoRA[16] (Low-Rank Adaptation) – метод обучения модели, который замораживает (не обучает) веса предобученной модели и обучает только небольшие низкоранговые матрицы обновлений, что позволяет существенно сократить количество обучаемых параметров и требования к хранению при сохранении полной скорости инференса.

LoRA решает проблему высокой стоимости тонкой настройки всё более крупных моделей на основе трансформеров, при которой обновление всех параметров требует огромных ресурсов GPU. Вместо изменения исходной матрицы весов W_0, LoRA представляет адаптацию ΔW как произведение двух значительно меньших матриц, используя тот факт, что эффективные обновления лежат в низкоразмерном подпространстве.

Конкретно, для полносвязанного слоя с $W_0 \in \mathbb{R}^{d\times k}$, LoRA вводит матрицу $\Delta W = B\,A$, где $A \in \mathbb{R}^{r\times k}, \quad B \in \mathbb{R}^{d\times r}, \quad r \ll \min(d,k)$. Обучаются только A и B (инициализируемые так: $A \sim \mathcal{N}(0,\sigma^2)$, $B = 0$), в то время как W_0 остаётся неизменным. Скалярный множитель $\frac{\alpha}{r}$ масштабирует обновление для стабилизации обучения. Во время работы матрица считается как $h = W_0\,x + (B\,A)\,x$.

LoRA совместим с другими методами повышения эффективности: в отличие от адаптеров, добавляющих новые слои, или prompt-tuning, расширяющего входную последовательность, он не увеличивает вычислительную сложность и не снижает максимальную длину обрабатываемых последовательностей.

1.5 Методы классификации текста

1.5.1 Logistic Regression

Логистическая регрессия — это статистический метод, используемый для моделирования вероятности двоичного исхода (например, успех/неудача) на основе одного или нескольких предикторов. Она преобразует линейную комбинацию признаков через логистическую (сигмоидную) функцию

$$\sigma(z) = \frac{1}{1+e^{-z}},$$

гарантируя, что предсказанные значения лежат между 0 и 1 и могут интерпретироваться как вероятности. В этой модели логарифм отношения шансов («логит») задаётся линейно:

$$\log \frac{\Pr(Y=1 \mid \mathbf{x})}{\Pr(Y=0 \mid \mathbf{x})} = \beta_0 + \sum_{i=1}^{p} \beta_i x_i.$$

Параметры оцениваются методом максимального правдоподобия: выбираются такие коэффициенты, которые максимизируют вероятность наблюдать имеющиеся данные при заданной модели. Так как функция логправдоподобия выпукла относительно коэффициентов, алгоритмы вроде метода Ньютона или градиентного подъёма надёжно сходятся к глобальному оптимуму. Оценка коэффициента β_i интерпретируется так: при увеличении x_i на единицу шансы наступления события умножаются на $\exp(\beta_i)$. Для классификации новых наблюдений вычисляют сигмоиду от линейного выражения и применяют порог (обычно 0.5): выше — класс «1», ниже — класс «0».

Логистическая регрессия ценится за простоту, интерпретируемость и способность работать как с непрерывными, так и с категориальными признаками. Она выступает надёжным базовым методом в задачах классификации — от медицинской диагностики до прогнозирования оттока клиентов в маркетинге — и её эффективность оценивается такими метриками, как точность, precision/recall, F1-мера и ROC-AUC. Главный недостаток модели — предположение о линейной зависимости между предикторами и логарифмом шансов; при его нарушении можно добавить перекрёстные и полиномиальные признаки или обратиться к более гибким методам.

1.5.2 ML-KNN

ML-kNN[17](многометочный k-ближайших соседей) — это ленивый алгоритм, расширяющий традиционный kNN для задач многометочной классификации. В многометочной постановке каждый объект может принадлежать нескольким категориям одновременно. ML-kNN предсказывает набор меток для нового объекта, анализируя его ближайших соседей в обучающей выборке и применяя вероятностное правило принятия решения на основе статистики совместного появления меток.

1. Представление меток и подсчет вхождений

   Пусть $Y = 1, 2, \ldots, Q$ — множество всех возможных меток. Каждый объект x представлен бинарным вектором категорий, где $y_x(l) = 1$, если метка l принадлежит x, и 0 в противном случае. Для данного x ML-kNN находит k ближайших соседей $N(x)$ и строит вектор подсчёта вхождений

C_x, чей l-й компонент вычисляется как

$$\tilde{C}_x(l) = \sum_{a \in N(x)} \tilde{y}_a(l)$$

2. Оценка априорных и апостериорных вероятностей (этап обучения)
   На этапе обучения ML-kNN рассматривает каждую метку l независимо и оценивает:
   - Априорные вероятности $P(H_l^1)$ и $P(H_l^0) = 1 - P(H_l^1)$, где H_l^1 обозначает событие, что случайный объект имеет (не имеет) метку l.
   - Условные вероятности $P(E_l^j \mid H_l^b)$, где E_l^j — событие того, что ровно j из k соседей имеют метку l, а $b \in \{0, 1\}$.
3. Предсказание по следующему правилу:
   Для каждого тестового объекта t ML-kNN сначала находит K ближайших соседей $N(t)$ в обучающей выборке. Пусть H_l^1 — событие, что t имеет метку l, а H_l^0 — событие, что t не имеет метки l. Обозначим E_l^j ($j \in \{0, 1, \ldots, K\}$) событие, что среди K ближайших соседей t ровно j объектов имеют метку l. Тогда на основе вектора подсчёта вхождений $\tilde{C}_t$ вектор категорий $\tilde{y}_t$ определяется по принципу:

$$\tilde{y}_t(l) \;=\; \arg\max_{b \in \{0,1\}} P\big(H_l^b \mid E_l^{\tilde{C}_t(l)}\big), \quad l \in Y.$$

4. Ранжирование меток Помимо бинарного предсказания y_t, ML-kNN вычисляет вещественный вектор ранжирования r_t, где для каждой l:

$$\tilde{r}_t(l) = P\big(H_l^1 \mid E_l^{\tilde{C}_t(l)}\big)$$

   Это ранжирование позволяет отбирать метки по порогу.

1.5.3 DNNC

Discriminative Nearest Neighbor Classification[18] (DNNC) реализуется как попарная функция соответствия: входное высказывание пользователя и эталонный пример соединяются в единую последовательность и обрабатываются с помощью BERT-подобной модели. На выходе текстовый векторы преобразуются с помощью функции, которая оценивает вероятность совпадения намерений пары. Во время работы выбирается эталон с

максимальным значением вероятности, после чего применяется порог для разграничения известных намерений и OOS-запросов.

Для снижения вычислительной нагрузки при большом количестве эталонных примеров предложен двухэтапный «совместный» (joint) механизм: сначала применяется более лёгкий метод отбора для выбора кандидатов, далее глубокая попарная модель DNNC доранжирует только отобранный набор. Данный приём сохраняет высокую дискриминативную способность при существенно уменьшенных требованиях к времени обработки.

1.5.4 CatBoost

CatBoost[7; 19] — это библиотека градиентного бустинга над решающими деревьями, которая изначально поддерживает работу с категориальными признаками без обширной предварительной обработки. В отличие от традиционных реализаций градиентного бустинга, CatBoost использует такие техники, основанныt на пермутационной статистике для предотвращения утечки целевых значений, и симметричные (обоюдные) деревья для снижения переобучения и повышения как стабильности, так и вычислительной эффективности.

При обработке текстовых признаков CatBoost использует многоступенчатый алгоритм, преобразующий строки в числовые векторы, пригодные для деревьев градиентного бустинга. Сначала текстовые столбцы загружаются, после чего каждая запись разбивается на токены — слова, символы или настраиваемые n-граммы. Затем строится словарь, в котором каждому уникальному токену присваивается числовой идентификатор. Каждая текстовая запись преобразуется в последовательность и передаётся на вход другим алгоритмам, которые вычисляют числовые сводки — индикаторы наличия токенов, условные вероятности по классам или оценки релевантности. Полученные признаки интегрируются в стандартный процесс обучения CatBoost.

Для признаков-эмбеддингов, представленных в виде фиксированных числовых векторов, CatBoost также генерирует скалярные признаки перед обучением деревьев. После указания таких столбцов поддерживаются два основных метода обработки. Линейный дискриминантный анализ (LDA) проецирует эмбеддинги в пространство низкой размерности и вычисляет для каждого класса значения гауссовой функции правдоподобия

(для классификации), а метод ближайших соседей (KNN) определяет ближайшие векторы из тренировочного набора, подсчитывая вхождения по классам или усредняя целевые значения соседей (для регрессии или классификации). Такие признаки, учитывающие информацию о классах или целевых значениях, позволяют CatBoost эффективно использовать семантику эмбеддингов без прямой работы с высокоразмерными координатами — хотя сами векторы можно добавить как обычные числовые признаки при необходимости.

1.6 Методы поиска текста

Поиск сходства векторов стал одной из ключевых операций в современных системах ИИ, когда самые разные данные — от слов и предложений до изображений и взаимодействий пользователей с контентом — отображаются в высокоразмерные эмбеддинги, в которых геометрическая близость отражает семантическое сходство. Это требует разработки высокоэффективных алгоритмов, способных обеспечить баланс между точностью, скоростью и объемом требуемой памяти. В частности, методы аппроксимационного поиска ближайших соседей (ANNS) стали незаменимыми в тех сценариях, где точный перебор оказывается неприемлемо затратным по времени.

Faiss[20] представляет собой набор инструментов, который сосредоточен исключительно на ядре ANNS: он не занимается извлечением эмбеддингов и не предоставляет сервисы управления базами данных, такие как транзакции или планирование запросов. Вместо этого Faiss предлагает богатый набор индексирующих примитивов с настраиваемыми параметрами, которые можно комбинировать, создавая специализированные алгоритмы поиска. Начиная от простых плоских индексов и заканчивая сложными многоступенчатыми структурами, Faiss позволяет пользователям оптимизировать решение под свои требования по скорости, точности и ресурсам.

Для быстрого поиска по большим коллекциям векторов Faiss реализует две взаимодополняющие стратегии, не требующие полного перебора. Индексы с инвертированным файлом (IVF) группируют базу данных на настраиваемое число «списков» и при выполнении запроса обрабатывают лишь их часть; при этом остаточное (residual) кодирование после грубого

квантования повышает точность. Графовые методы, такие как Hierarchical Small Navigable World (HNSW)[21], строят навигируемые маломировые графы для эффективного поиска соседей.

1.7 Способы расширения данных

В работе [22] уделили внимание критическому недостатку систем диалогов с задачами: склонности классификаторов намерений к ошибкам при встрече с очень похожими текстами (hard-negatives) внеобласти (OOS) высказываниями, которые похожи на поддерживаемые интенты, но на самом деле выходят за рамки домена системы. Авторы представляют полностью автоматизированный алгоритм на базе ChatGPT: сначала выделяют слова для каждого интента, затем генерируют OOS-примеры, включающие эти слова, и на последнем шаге с помощью двухступенчатой проверки GPT убеждаются, что полученные высказывания действительно не соответствуют ни одному поддерживаемому интенту. Применив этот подход к пяти наборам данных, они сформировали 3 732 таких высказываний. При оценке оказалось, что модели, обученные только на доменных данных, слишком самоуверенны на этих похожих примерах, но включение сгенерированных высказываний в тренировочный набор резко улучшает метрики.

Переписать немного обращения к работам В работе Hu, Khosmood, Edalat [23] схожим образом возвращаются к задаче классификации намерений без дополнительного обучения, используя текстовые эмбеддинги, чтобы обойтись без каких-либо размеченных примеров. Они предлагают несколько схем дополнения простого подхода косинусного сходства описаниями интентов — короткими декларированиями, сохраняющими ключевые слова из названий интентов (например, «BookRestaurant» превращается в «пользователь хочет забронировать столик в ресторане»).

Для автоматизации расширерния данных можно использовать библиотеку DSPy[24] (Declarative Self-improving Python). Она представляет собой Python-фреймворк для декларативного описания взаимодействия с языковыми моделями и их автоматической оптимизации. В отличие от традиционных подходов, где разработчик вручную конструирует многослойные шаблоны промптов, dspy формирует граф текстовых преобразований,

в котором каждый узел задаётся через формальную сигнатуру входов и выходов.

Архитектура dspy опирается на три центральные абстракции:

1. Сигнатуры, определяющие контракт модуля путём спецификации типов и форматов входных и выходных параметров;
2. Модули, инкапсулирующие распространённые техники промптинга и работу с внешними инструментами (`Chain-of-Thought`, `few-shot` и другие) в виде параметризуемых компонентов;
3. Телепромптеры (teleprompters), автоматически подбирающие демонстрации и инструкции на основе набора «учебных» примеров и заданной метрики, а при необходимости оптимизирующие параметры модели.

Оптимизационный процесс dspy заключается в итеративном исполнении примеров через исходный конвейер (режим), сборе успешных траекторий работы модулей и отборе наиболее эффективных демонстраций и инструкций. По результатам этой фазы возвращается оптимизированная декларативная программа, готовая к промышленному использованию.

1.8 Метрики для оценки качества алгоритма

1.8.1 Метрики поиска

Эти метрики используются для оценки качества систем поиска и рекомендаций, которые возвращают ранжированный список документов или элементов. Поскольку пользователям важнее получить релевантные ответы на первых позициях, метрики ранжирования показывают, насколько хорошо система выводит нужные объекты вверху. С их помощью можно сравнивать разные алгоритмы, подбирать оптимальные параметры и отслеживать прогресс при обучении моделей.

1. Precision@k

   Precision@k показывает, какую долю из первых k результатов составляют релевантные документы:

$$P@k = \frac{1}{k} \sum_{i=1}^{k} \mathrm{rel}_i,$$

где rel_i равно 1, если документ на позиции i релевантен, и 0 — иначе. Эта метрика проста и интуитивно понятна, что является её сильной стороной: она прямо отражает практическую пользу выдачи при просмотре первых k ответов. Однако P@k игнорирует порядок внутри первых k (то есть один релевантный документ на 1-й позиции и на k-й считаются одинаковыми) и полностью не учитывает результаты после k-го, что может приводить к переоценке алгоритмов, которые хорошо работают только на небольшом числе верхних позиций.

2. NDCG@k (Normalized Discounted Cumulative Gain)
   NDCG@k учитывает и степень релевантности (градуированную оценку), и штрафует более низкие позиции:

$$\mathrm{DCG}_k = \sum_{i=1}^{k} \frac{2^{\mathrm{rel}_i} - 1}{\log_2(i+1)}, \qquad \mathrm{NDCG}_k = \frac{\mathrm{DCG}_k}{\mathrm{IDCG}_k},$$

   где IDCG_k – максимальное возможное значение DCG при идеальном ранжировании. Благодаря учёту логарифмического дисконтирования NDCG снижает вклад документов, появившихся дальше, а использование $2^{\mathrm{rel}_i} - 1$ усиливает вклад особо релевантных материалов. Это делает NDCG гибкой и информативной: она отражает разницу между «очень» и «слабо» релевантными документами, но одновременно более сложна в вычислении и интерпретации, чем P@k, и требует наличия градуированных меток релевантности.

3. MAP (Mean Average Precision)
   MAP усредняет точность с учётом позиций всех релевантных документов и затем берёт среднее по запросам. Сначала для каждого запроса вычисляют

$$\mathrm{AP} = \frac{1}{R} \sum_{i=1}^{n} P@i \ \mathrm{rel}_i,$$

   где R – общее число релевантных документов для запроса, а n – рассматриваемая длина выдачи. Затем

$$\mathrm{MAP} = \frac{1}{|Q|} \sum_{q \in Q} \mathrm{AP}_q.$$

MAP хорошо отражает ранжирование в целом, поскольку чем раньше появляются релевантные, тем выше значение AP, и при этом учитываются все такие документы. Однако она не подходит для градуированных оценок и зависит от того, сколько релевантных документов существует и до какого n мы считаем выдачу, что усложняет сравнение моделей на разных наборах данных.

4. MRR (Mean Reciprocal Rank)

   MRR показывает, как быстро в среднем находится первый релевантный документ. Для каждого запроса берут обратную величину ранга первого релевантного результата rank_q:

$$\mathrm{RR}_q = \frac{1}{\mathrm{rank}_q}, \qquad \mathrm{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \mathrm{RR}_q.$$

   Эта метрика отличается простотой и прозрачностью: она сразу показывает, на какой позиции в среднем появляется первый релевантный ответ. С другой стороны, MRR игнорирует все релевантные документы после первого, поэтому не отражает полноту выдачи и может быть неинформативна, если для пользователя важны не только первые найденные, но и последующие релевантные результаты.

1.8.2 Метрики классификации

Метрики, описанные в данном пункте, применяются при оценке классификаторов и помогают понять, насколько точно модель определяет положительный и отрицательный классы, а также насколько она сбалансирована при разных соотношениях классов. С их помощью можно выбирать лучшее пороговое значение и сравнивать алгоритмы.

Для наглядного представления результатов классификации служит матрица ошибок (confusion matrix), в которой по строкам указаны предсказания модели $f(x)$, а по столбцам — истинные значения y. Эта таблица позволяет сразу увидеть, сколько примеров модель правильно и неправильно классифицировала:

	$Y = 0$ (Отрицательный)	$y = 1$ (Положительный)
$f(x) = 0$	TN	FN
$f(x) = 1$	FP	TP

Где

- TN (True Negative) – модель правильно предсказала отрицательный класс;
- FN (False Negative) – модель ошибочно отнесла положительный пример к отрицательному;
- FP (False Positive) – модель ошибочно отнесла отрицательный пример к положительному;
- TP (True Positive) – модель правильно предсказала положительный класс.

На основе элементов матрицы ошибок можно вычислить ряд ключевых метрик:

1. Accuracy (доля верных классификаций)

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}},$$

где TN (true negatives) – число правильно определённых отрицательных примеров. Accuracy отражает общую долю правильных ответов модели и проста для интерпретации, однако на сильно несбалансированных данных она может вводить в заблуждение: модель, предсказывающая всегда «отрицательный», при 99 % отрицательных примерах получит 99 % точности, хотя фактически будет бесполезна.

2. Precision (точность предсказания положительного класса)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$

где TP (true positives) – число верно предсказанных положительных примеров, а FP (false positives) — количество ложно «положительных». Precision показывает, какую долю среди предсказанных моделью «положительных» примеров составляют действительно положительные. Это важно, когда ложные срабатывания дорого обходятся (например, спам-фильтр не должен блокировать важные письма). При этом Precision игнорирует все пропущенные положительные примеры (FN), поэтому модель, слишком консервативно отмечающая положительные случаи, может иметь высокий Precision при очень низком Recall.

3. Recall (полнота, чувствительность)

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

где FN (false negatives) – число пропущенных моделью положительных примеров. Recall показывает, какую долю от всех истинно положительных примеров модель смогла обнаружить, что актуально, когда важно не упустить ни одного положительного случая (например, при диагностике заболеваний). Достоинство этой метрики — фокус на захват всех «плюсов», однако она не учитывает ложно положительные срабатывания, и высокая Recall может достигаться ценой большого числа FP.

4. F1-score

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

F1 объединяет точность и полноту, отдавая больше веса тем случаям, когда одна из метрик низка, и тем самым обеспечивает сбалансированную оценку работы модели при неоднородных классах. Это полезно, когда важно одновременно и не пропускать положительные примеры, и не допускать много ложных срабатываний. Однако F1 не учитывает TN и потому не отражает способность модели правильно распознавать отрицательные примеры; кроме того, оно предполагает равный вес Precision и Recall, что не всегда соответствует бизнес-целям.

5. ROC AUC

ROC-кривая строится по точкам (FPR, TPR), где

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}},$$

а AUC – это интеграл под этой кривой. Высокий ROC AUC означает, что модель хорошо различает положительные и отрицательные примеры при любом пороге, что делает её независимой от выбора порога и удобной для сравнения алгоритмов. С другой стороны, при сильном дисбалансе классов AUC может давать искажённо высокую оценку, поскольку учитывает весь диапазон порогов, включая нерелевантные для прикладных задач точки, и не показывает, как модель ведёт себя при конкретных настройках.

2 Реализация

2.1 Архитектура

В рамках этой работы построен фреймворк AutoIntent для классификации намерений и состоит из последовательности из трех основных типов узлов: Embedding (векторные представления), Scoring (оценка) и Decision (оценка качества). Каждый узел отвечает за свой этап обработки: например, узел кодирования преобразует входные реплики в векторные представления, узел оценивания вычисляет оценки принадлежности различным намерениям, а узел принятия решения на основе этих оценок определяет итоговое намерение. Такая многоуровневая архитектура позволяет разложить сложную задачу на отдельные компоненты с чётко определёнными ролями. Все узлы объединены в объект Pipeline, который управляет их совместной работой и оптимизацией.

Важной особенностью архитектуры AutoIntent является модульность и взаимозаменяемость компонентов. Для каждого узла определён набор возможных модулей (алгоритмов) и метрик качества. Базовые классы (*BaseModule*, *BaseEmbedding*, *BaseScorer*, *BaseDecision*) задают единый интерфейс для модулей каждого типа, что упрощает добавление новых методов.

Архитектура AutoIntent также нацелена на повторяемость экспериментов и воспроизводимость результатов. Для этого все параметры и выбор модулей описываются в едином конфигурационном файле (в формате yaml или json), из которого инициализируется оптимизационный конвейер. Конфигурация полностью определяет состав узлов, что позволяет запустить идентичный эксперимент на других данных или средах. В процессе работы конвейера используется объект контекста (Context), который хранит текущее состояние – данные, разделённые на обучающие/тестовые выборки, лучшие найденные параметры, промежуточные метрики и обеспечивает взаимодействие между узлами. Ниже представлена архитектура на рисунке 2.1.

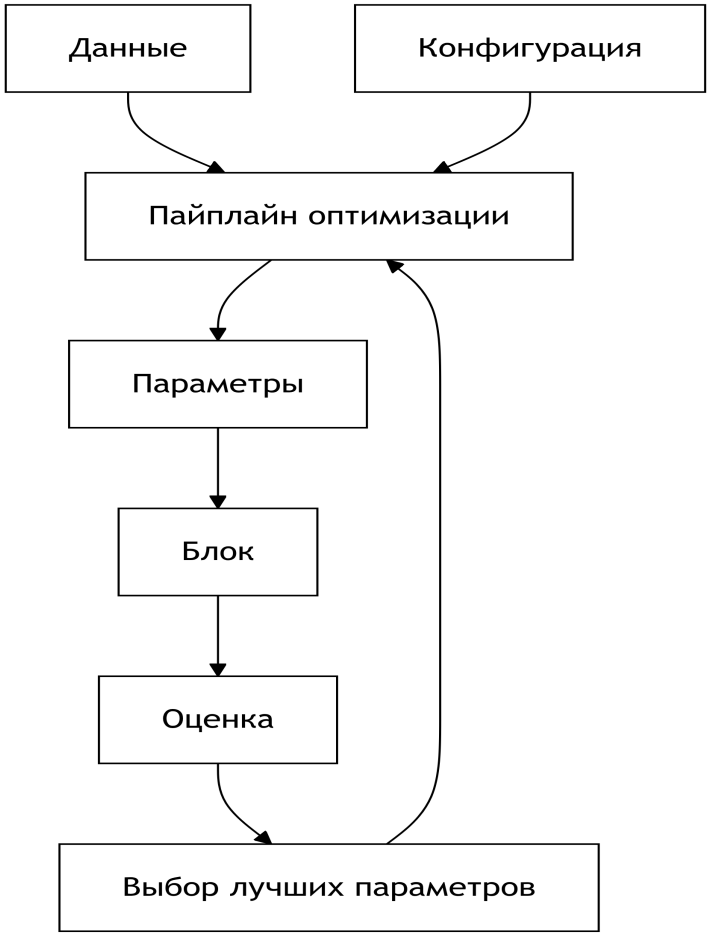

Рисунок 2.1 — Схема фреймворка

2.2 Управление данными

Данные в AutoIntent организованы в стандартизованный формат и обрабатываются через специализированный класс Dataset. Исходный набор данных представляет собой JSON или репозиторий на HuggingFace с разбивкой на сплиты (например, train, validation, test), где каждая запись содержит текст запроса пользователя (utterance) и метку интента (label). Класс Dataset загружает эти данные и приводит к внутреннему представлению на базе HuggingFace Dataset. При инициализации Dataset автоматически собирается список всех интентов и их описаний, а также проверяется, является ли задача многоклассовой или многолейбловой (мультиметочной).

Для эффективной работы с данными *AutoIntent* использует компонент *DataHandler*, который берёт на себя разделение данных на обучающие и проверочные части согласно настройкам. Параметры разделения задают-

ся в конфигурации *DataConfig*. Например, поле *scheme* определяет схему валидации: значение `ho` соответствует классическому отложенному набору (*hold-out*), а `cv` – кросс-валидации. В случае *hold-out*, если входной датасет не содержит явного валидационного-сплита, то *DataHandler* автоматически выделит из обучающих данных долю данных (по умолчанию 20%) для валидации. Также DataHandler поддерживает режим *few-shot*, когда для обучения используется лишь несколько примеров каждого класса – это бывает полезно при несбаланированом распределении классов. При необходимости DataHandler дополнительно разделяет обучающую выборку между узлами scoring и decision, чтобы избежать утечки данных. При заданном $\text{separation}_{\text{ratio}}$ часть обучающих примеров резервируется для обучения финального классификатора, а остальные – для подбора промежуточных модулей.

Если задана кросс-валидация, *DataHandler* объединяет все обучающие данные и производит стратифицированное разбиение на N под наборов данных (число наборов данных задаётся параметром n_{folds}, по умолчанию 3). В результате, данные в содержат разбиение на либо train/validation (для hold-out), либо наборы $train_0, \ldots, train_{(N-1)}$ для кросс-валидации, а также необязательный test (если пользователь предоставил тестовую выборку).

2.3 Конфигурация

Весь процесс оптимизации в AutoIntent управляется единым конфигурационным описанием, что гарантирует повторяемость настройки эксперимента и удобство его запуска. Конфигурация задаётся пользователем в виде YAML- или JSON-файла, либо соответствующего Python-словаря, который затем валидируется и загружается с помощью Pydantic-моделей. Основная модель конфигурации – класс *OptimizationConfig* – включает в себя несколько вложенных конфигураций для данных (*DataConfig*), логирования (*LoggingConfig*), параметров моделей (*EmbedderConfig, CrossEncoderConfig, HFModelConfig*), которые будут использоваться если подобные модели не оптимизировалась в рамках подбора параметров, а главное – описание поискового пространства модулей ($\text{search}_{\text{space}}$). Структура $\text{search}_{\text{space}}$ представляет собой список узлов, для каждого из которых указаны: тип узла ($\text{node}_{\text{type}}$), целевая метрика оп-

тимизации ($target_{metric}$) и список модулей с набором их гиперпараметров ($search_{space}$ внутри узла). Например, в конфигурации можно задать узел типа «scoring» с целевой метрикой F1, в котором рассматриваются два модуля – k-NN с различными значениями k и линейный классификатор с параметрами по умолчанию (см 2.1). Таким образом, конфигурация полностью описывает, какие комбинации методов будут перебраны фреймворком.

```yaml
- node_type: scoring
  target_metric: scoring_f1
  search_space:
    - module_name: knn
      k:
        low: 1
        high: 20
    - module_name: linear
- node_type: decision
  target_metric: decision_accuracy
  search_space:
    - module_name: argmax
    - module_name: jinoos
```

Листинг 2.1 – Пример конфигурации

При запуске оптимизации конфигурационный файл считывается и инициализируется объект OptimizationConfig. Далее, на основе полученного конфигурационного объекта инициализируется сам конвейер: создаются оптимизаторы узлов (NodeOptimizer) для каждого описанного узла в поисковом пространстве, а также устанавливаются глобальные параметры (random seed, метод выборки гиперпараметров и так далее).

Пользователь может хранить различные конфигурации для разных датасетов или сценариев и запускать их без изменения кода, просто указывая нужный файл. Каждый эксперимент при запуске сохраняет свою конфигурацию вместе с результатами, что дополнительно гарантирует прозрачность – всегда можно посмотреть, с какими именно параметрами получена та или иная модель.

Для валидации конфигурации используется библиотека Pydantic, которая позволяет задавать типы и ограничения для каждого поля. DatasetConfig, LoggingConfig и другие вложенные конфигурации наследу-

ются от базового класса BaseModel, что позволяют делать проверку корректности сразу.

Для проверки конфигурации модулей во время запуска программы генерируются Pydantic классы для каждого из модулей. Для этого у каждого модуля считываются аннотации типов из метода `from_context`, через который все модули инициализируются во время оптимизации. Для более точной валидации были добавлены аннотации `PositiveInt` и `FloatFromZeroToOne` для параметров, которые позволяют задать только положительные значения и значения от 0 до 1 соответственно. Это позволяет избежать ошибок при передаче некорректных значений в модули, которые могут привести к сбоям во время работы. Также для каждого числового типа можно задать диапазон значений, что позволяет ограничить поиск только теми значениями (`low`, `hight`) и шаг для поиска (`step`) и `Literal` для категориальных параметров.

Также для улучшения опыта пользователя генерируется json-схема (JSON Schema) для каждого конфигурационного файла, что позволяет использовать автозаполнение в IDE и проверку типов. Это позволяет избежать ошибок при написании конфигурации и ускоряет процесс настройки эксперимента. Например, если пользователь попытается указать строку вместо числа для параметра k в конфигурации, то IDE выдаст ошибку.

2.4 Модули

Каждый узел конвейера AutoIntent может быть реализован различными алгоритмическими модулями, что отражает принцип модульности и расширяемости фреймворка. Рассмотрим основные категории узлов и соответствующие им модули.

2.4.1 Encoder

Узел Embedding представляет отвечает за выбор векторизации текста и выполняет промежуточную оценку векторных представлений через два доступных модуля:

- *RetrievalAimedEmbedding* – оценивает векторное представление по тому, насколько хорошо оно группирует примеры одного намерения, используя тест k-NN поиска. Он индексирует обучающие примеры в векторном пространстве (с использованием выбранной модели векторизации) и для

каждого выражения из валидационного набора проверяет, содержат ли его k ближайших соседей правильную метку намерения. Вычисляются такие метрики, как recall, ndcg, map. Более высокий балл указывает на то, что векторизация группирует схожие намерения рядом, что является желаемым для дальнейшей классификации.

- *LogregAimedEmbedding* – оценивает векторное представление путем обучения простого классификатора логистической регрессии на векторных представлениях. Он использует модель векторизации для преобразования всех обучающих выражений в векторные признаки, затем обучает логистическую регрессию (опционально с использованием внутренней кросс-валидации для повышения устойчивости). Точность на валидационном наборе (или другая выбранная метрика) этого классификатора отражает, насколько линейно разделимы классы намерений в этом векторном пространстве. Это дает прямой сигнал о полезности векторизации для классификации.

Отделяя выбор эмбеддера, AutoIntent избегает повторного обучения тяжелых моделей для каждой попытки классификатора — сначала он находит лучшее текстовое представление (с использованием легких тестов), а затем переиспользует его.

2.4.2 Scoring

Scoring является главным узлом конвейера, включающим модели машинного обучения, которые непосредственно классифицируют высказывания в намерения. AutoIntent предоставляет богатую коллекцию модулей оценки, каждый из которых реализует различный алгоритм или подход к моделированию для классификации. Во время AutoML оптимизатор оценки будет перебирать различные комбинации этих модулей и их гиперпараметров, чтобы найти лучший вариант. Ключевые модули оценки включают:

- *KNNScorer* — классификатор на основе метода k ближайших соседей, использующий векторные эмбеддинги. Он использует выбранный эмбеддер для векторизации всех обучающих выражений и хранит их в VectorIndex для эффективного поиска ближайших соседей. Во время предсказания он находит k ближайших обучающих примеров к запросу и вычисляет оценки классов, используя стратегии, такие как равное голосование или метод «ближайшего класса» (каждому классу присваивается кредит от

его самого близкого соседа). Это быстрый, непараметрический метод, который часто служит сильной базовой моделью. Его основные гиперпараметры — это k (число соседей) и схема взвешивания, и он зависит от эмбеддера (из узла Embedding) для подачи векторного представления.

- *LinearScorer* – линейная модель, которая обучает веса для признаков, чтобы классифицировать намерения. На практике этот модуль использует логистическую регрессию для обучения на эмбеддингах выражений. Он может иметь гиперпараметры, такие как сила регуляризации. Это полезно как легковесная обучаемая модель — менее мощная, чем глубокие сети, но быстрая и иногда достаточная для простых задач.
- *SklearnScorer* – универсальный интерфейс для классификаторов scikit-learn (таких как SVM или случайные леса). Он позволяет легко подключать классические модели машинного обучения. Конкретный алгоритм может быть частью пространства поиска (например, можно попробовать SVM против Наивного Байеса, изменяя параметр).
- *BertScorer* – классификатор на основе трансформера с возможностью тонкой настройки. Этот модуль использует трансформер из HuggingFace с головой для классификации. Он будет тонко настраивать трансформер на задаче классификации намерений, создавая мощный классификатор, который часто достигает высокой точности, но требует большего времени на обучение. Гиперпараметры для BertScorer включают такие параметры, как название базовой модели, количество эпох, скорость обучения и так далее, которые AutoIntent может перебирать. Этот модуль олицетворяет нейронную часть спектра в поисковом пространстве.
- *LoRAScorer* и *PTuningScorer* – варианты тонкой настройки с параметрической эффективностью. BERTLoRAScorer применяет LoRA (Low-Rank Adaptation) для тонкой настройки трансформера с меньшим числом обучаемых параметров (внедрение обученных матриц адаптации) — это ускоряет обучение и снижает использование памяти. PTuningScorer, вероятно, относится к Prompt Tuning или Prefix Tuning, где обучается небольшая группа эмбеддингов подсказок, в то время как основная модель остается фиксированной. Эти модули позволяют AutoIntent исследовать методы тонкой настройки, которые эффективно используют большие модели на малых данных.

- *DescriptionScorer* – подход на основе семантического сходства, использующий описания намерений. Этот модуль использует естественные языковые описания каждого намерения и вычисляет сходство между входным выражением и текстами описаний намерений. Он может работать в двух режимах: в режиме би-энкодера, когда выражение и описание кодируются эмбеддером и сравниваются (например, с использованием косинусного сходства), и в режиме кросс-энкодера, когда пара выражение-описание подается в трансформерную модель (например, модель с перекрестным вниманием), чтобы напрямую вычислить оценку релевантности. В любом случае DescriptionScorer выводит оценку для каждого намерения на основе того, насколько хорошо выражение соответствует описанию намерения, а не на основе примеров из обучающего набора. Это мощный способ работы с намерениями с небольшим количеством примеров или с использованием семантических знаний. Гиперпараметры этого модуля включают выбор между би- или кросс-энкодингом и выбор моделей для каждого из них.
- *RerankScorer* – двухступенчатая гибридная модель, которая сочетает извлечение и повторную сортировку с помощью кросс-энкодера. Внутри RerankScorer наследует KNNScorer (проводит начальный поиск соседей), а затем применяет кросс-энкодер для повторной оценки топ-m соседних результатов. Получив запрос, он находит топ-кандидатов на основе KNN, а затем более точно оценивает этих кандидатов, подавая запрос и каждый пример кандидата в модель кросс-энкодера для получения уточненной оценки сходства. Это повышает точность за счет использования мощной модели на небольшой подмножестве кандидатов, что намного быстрее, чем использование кросс-энкодера для всех намерений или всех обучающих примеров. Гиперпараметры включают k (для начального извлечения), m (сколько из них нужно повторно отсортировать, если это отличается от k).

Все эти модули оценки выводят вектор оценок: для заданного выражения они генерируют оценку (или вероятность) для каждого класса намерений. Во время фазы AutoML каждый модуль оценивается путем обучения на обучающем наборе и вычисления одной или нескольких метрик на валидационном наборе через метод score(). Метрики могут включать точ-

ность, F1 и т.д., и AutoIntent определяет отдельные функции для метрик для многоклассовых и многозначных случаев, чтобы обеспечить соответствующую оценку. Контекст используется для подачи модулям правильных разделов данных и для сбора результатов. Лучший модуль оценки (с наивысшей метрикой) выбирается для включения в финальный пайплайн. В финальном пайплайне вывода выбранный модуль оценки оборачивается как InferenceNode и будет отвечать за производство оценок для новых выражений. Он взаимодействует с нодой Decision.

2.4.3 Decision

Узел решения (Decision) – финальный узел, принимающий решение на основе результатов предыдущих шагов. Его задача – вынести окончательный вердикт: один (или несколько) интентов, которые соответствуют запросу пользователя, либо определить, что запрос не относится ни к одному из известных интентов (out-of-scope). Здесь тоже предусмотрено несколько стратегий: Decision является последним этапом пайплайна, который принимает необработанные оценки из нода Scoring и преобразует их в окончательные предсказанные метки для каждого выражения. Этот этап особенно важен для применения порогового отклонения (обнаружение OOS) или выбора нескольких намерений в многозначных сценариях. AutoIntent предлагает несколько модулей принятия решений, все из которых наследуют от BaseDecision:

- *ArgmaxDecision* – самая простая стратегия, которая всегда выбирает намерение с наивысшей оценкой в качестве предсказания. Этот модуль выполняет $\text{prediction} = \arg\max score_j$ для каждого выражения. Он не поддерживает OOS (так как всегда выбирает одно из намерений) и используется только для многоклассовой классификации. Этот модуль часто является лучшим выбором, когда OOS не требуется, а оценки классификатора являются надежными вероятностями.
- *ThresholdDecision* – конфигурируемый предсказатель на основе порога, который может работать как для многоклассовой, так и для многометочной классификации. Для многоклассовой классификации он выбирает топовое намерение, если его оценка больше некоторого порога t, в противном случае выводит OOS (или «нет намерения»). В многометоч-

ной режиме он может применить порог для каждого класса (или вектор порогов), чтобы решить, какие намерения включить в предсказание.

- *TunableDecision* – более универсальный модуль оптимизации порога, который работает как для многоклассовой, так и для многометочной классификации и использует Optuna для настройки. В TunableDecision, вместо того чтобы сканировать фиксированную сетку порогов, исследуется набор пороговых значений (один глобальный порог или по одному для каждого класса, в зависимости от сценария), которые максимизируют выбранную целевую метрику. Это эффективно находит оптимальные пороговые значения для каждого класса или в целом, что может значительно улучшить баланс между точностью и полнотой в многометочных задачах или корректно настроить OOS в многоклассовых задачах. TunableDecision поддерживает OOS потенциально обучая порог, который заставляет некоторые низкие оценки восприниматься как отсутствие намерения.
- *AdaptiveDecision* – модуль, предназначенный для многозначной классификации с «адаптивной» стратегией пороговой настройки. Многозначные задачи часто требуют выбора порога для каждого класса, а иногда и их корректировки в зависимости от того, сколько меток нужно предсказать. AdaptiveDecision использует более простой подход, чем полная настройка Optuna: он применяет поиск по коэффициенту масштабирования r, который применяется к стандартным порогам, чтобы оптимизировать метрику (например, F1). Фактически, он может начать с того, что порог каждого класса устанавливается на стандартное значение (например, 0.5) или на основе статистики обучающего набора, а затем находит единственный множитель r, который наилучшим образом регулирует все пороги. AdaptiveDecision не обрабатывает OOS явно. Этот модуль полезен, когда ожидается переменное количество меток и нужно настроить чувствительность.

Во время оптимизации пайплайна модули Decision оцениваются после того, как выбран модуль оценки. На практике, лучшие результаты валидации выбранного модуля Scoring (для удержания данных или кросс-валидации) подаются в каждый кандидатный модуль Decision, чтобы проверить, насколько хорошо он преобразует оценки в правильные предсказа-

ния. Например, после того как классификатор фиксируется, AutoIntent может симулировать различные стратегии порогов на его выходных данных, чтобы увидеть, какой из них даст наибольшую точность или F1 на валидации. Методы score() модулей Decision обрабатывают эту оценку, внутренне вызывая метод predict() на векторах валидационных оценок и сравнивая их с истинными метками. Целевая метрика для стадии принятия решения может быть такой же, как и для общей метрики (например, если оптимизируется точность пайплайна, то будет выбран модуль принятия решения, который дает наибольшую точность на валидации). После выбора модуль принятия решения сохраняется как последняя часть пайплайна. Во время вывода модуль принятия решения просто принимает вектор оценок, выведенный модулем Scoring, и генерирует окончательный набор меток. Он также может вывести дополнительные метаданные (например, оценки или теги), если это требуется через специальный вызов, но в целом его выходом является только предсказанное намерение(я). Эта модульная структура позволяет легко изменять, насколько консервативной или агрессивной будет система при предсказании намерений против OOS, просто изменяя модуль принятия решения (без повторного обучения классификатора).

2.5 Обучение

Оптимизация в контексте AutoIntent представляет собой процесс автоматического подбора оптимальной комбинации модулей и их гиперпараметров для заданного набора данных. Вначале конвейер проверяет, не находится ли он в режиме инференса (то есть, чтобы обучающий метод не был вызван на уже готовом предсказательном конвейере), затем формирует новый Context и настраивает его (данные, логирование, модели если заданы). Далее происходит цикл оптимизации по узлам: AutoIntent проходит последовательно узел за узлом в определённом порядке (сначала embedding, потом scoring и в конце decision), запускает для каждого соответствующий NodeOptimizer и ждёт, пока он подберёт лучший модуль для своего этапа.

Процесс внутри NodeOptimizer построен на использовании библиотеки Optuna[25] для эффективного исследования пространства параметров. Каждому узлу соответствует список модулей-кандидатов (из конфи-

гурации $\text{search}_{\text{space}}$), и для каждого модуля задан диапазон значений гиперпараметров. Затем оптимизатор осуществляет вложенный цикл: перебирает модули по очереди и для каждого выполняет серию испытаний с разными комбинациями гиперпараметров. Методология перебора задаётся параметром sampler – AutoIntent поддерживает несколько стратегий: `brute` (полный перебор всех комбинаций, если их число невелико), `random` (случайный поиск фиксированного числа итераций) и `tpe` (алгоритм Tree-Structured Parzen Estimator из Optuna для байесовской оптимизации). По умолчанию используется brute-force, однако для непрерывных или обширных пространств целесообразно переключиться на более умные подходы.

Внутри одного испытания NodeOptimizer выполняет следующие шаги. Сначала на основе текущих значений гиперпараметров создаётся конфигурация модуля через Optuna, которая подбирает значения в зависиомости от стратегии. Затем создаётся экземпляр класса модуля с заданными параметрами, при необходимости дополняются имплицитные параметры (в некоторых модулях, например, может вычисляться размер выходного слоя на основе данных), и модуль тренируется/применяется на обучающих данных из Context и выдает предсказания на валидационном наборе. Целевое значение извлекается и возвращается Optuna, которая на основании него принимает решение о выборе следующей комбинации гиперпараметров. Параллельно, вся информация о проведённом испытании фиксируется. AutoIntent логирует параметры модуля, полученные метрики, возможные артефакты, а также немедленно сохраняет обновлённый прогресс на диск. Это позволяет даже в случае прерывания эксперимента иметь данные о уже проверенных комбинациях и, при надлежащих настройках, потом восстановиться и продолжить поиск.

После того, как все узлы оптимизированы, AutoIntent формирует финальный конвейер (inference). На этом шаге классы-оптимизаторы узлов (NodeOptimizer) заменяются на соответствующие им инференс-обёртки (InferenceNode), содержащие внутри уже настроенные лучшие модули. Конвейер обновляет свой список узлов: вместо множества пробуемых модулей фиксируется по одному выбранному модулю на узел.

2.6 Логирование

В процессе работы AutoIntent ведёт два уровня логов: системные (через стандартный модуль logging Python) и метрики эксперимента (через CallbackHandler). В системные логи выводятся информационные сообщения о ходе оптимизации, предупреждения о потенциальных некорректных настройках и прочие служебные сведения – они, как правило, сохраняются в текстовый файл в папке запуска. Метрики обрабатываются CallbackHandler: при старте каждого прогона модуля context.callback$_{\text{handler.start run}}$() открывает новый эксперимент в системе мониторинга, а далее для каждого эксперимента вызываются start$_{\text{module}}$, log$_{\text{metrics}}$ и так далее. В итоге, если подключен например Weights & Biases, пользователь в реальном времени видит графики метрик по ходу перебора.

3 Эксперименты

- Сравнение с фреймворкам по пресетам
- Сравнение модулей
- Сравнение энкодеров и промтов
- Сравнение few-shot
- Сравнение качества на аугментированных данных

Заключение

СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ

1. *Jones, Karen Sparck.* A Statistical Interpretation of Term Specificity and Its Application in Retrieval // Journal of Documentation. — 1972. — 1 янв. — Т. 28, № 1. — С. 11—21. — ISSN 0022-0418. — DOI: 10.1108/eb026526. — URL: https://doi.org/10.1108/eb026526.
2. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / J. Devlin [и др.]. — 24.05.2019. — DOI: 10.48550/arXiv.1810.04805. — arXiv: 1810.04805 [cs]. — URL: http://arxiv.org/abs/1810.04805 (дата обр. 02.08.2023). — Пред. пуб.
3. *LeDell E., Poirier S.* H2O AutoML: Scalable Automatic Machine Learning. — 2020.
4. LightAutoML: AutoML Solution for a Large Financial Services Ecosystem / A. Vakhrushev [и др.]. — 05.04.2022. — DOI: 10.48550/arXiv.2109.01528. — arXiv: 2109.01528 [cs]. — URL: http://arxiv.org/abs/2109.01528 (дата обр. 04.04.2025). — Пред. пуб.
5. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data / N. Erickson [и др.]. — 13.03.2020. — DOI: 10.48550/arXiv.2003.06505. — arXiv: 2003.06505 [stat]. — URL: http://arxiv.org/abs/2003.06505 (дата обр. 04.04.2025). — Пред. пуб.
6. *Chen T., Guestrin C.* XGBoost: A Scalable Tree Boosting System // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. — New York, NY, USA : Association for Computing Machinery, 13.08.2016. — С. 785—794. — (KDD '16). — ISBN 978-1-4503-4232-2. — DOI: 10.1145/2939672.2939785. — URL: https://dl.acm.org/doi/10.1145/2939672.2939785 (дата обр. 24.05.2023).
7. *Dorogush A. V., Ershov V., Gulin A.* CatBoost: Gradient Boosting with Categorical Features Support. — 24.10.2018. — URL: http://arxiv.org/abs/1810.11363 (дата обр. 28.05.2023).

8. Automated Evolutionary Approach for the Design of Composite Machine Learning Pipelines / N. O. Nikitin [и др.] // Future Generation Computer Systems. — 2022. — Февр. — Т. 127. — С. 109—125. — ISSN 0167739X. — DOI: 10.1016/j.future.2021.08.022. — arXiv: 2106.15397 [cs]. — URL: http://arxiv.org/abs/2106.15397 (дата обр. 18.05.2025).

9. Multi-Objective Evolutionary Design of Composite Data-Driven Models / I. S. Polonskaia [и др.]. — 17.05.2021. — DOI: 10.48550/arXiv.2103.01301. — arXiv: 2103.01301 [cs]. — URL: http://arxiv.org/abs/2103.01301 (дата обр. 18.05.2025). — Пред. пуб.

10. Multilingual E5 Text Embeddings: A Technical Report / L. Wang [и др.]. — 08.02.2024. — DOI: 10.48550/arXiv.2402.05672. — arXiv: 2402.05672 [cs]. — URL: http://arxiv.org/abs/2402.05672 (дата обр. 15.03.2025). — Пред. пуб.

11. Attention Is All You Need / A. Vaswani [и др.] // Advances in Neural Information Processing Systems. Т. 30. — 2017. — URL: https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (дата обр. 28.02.2023).

12. Language Models Are Unsupervised Multitask Learners / A. Radford [и др.]. — 2019. — URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

13. Deep Contextualized Word Representations / M. E. Peters [и др.]. — 22.03.2018. — URL: http://arxiv.org/abs/1802.05365 (дата обр. 28.02.2023).

14. *Reimers N.*, *Gurevych I.* Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks. — 27.08.2019. — URL: http://arxiv.org/abs/1908.10084 (дата обр. 20.12.2022).

15. GPT Understands, Too / X. Liu [и др.]. — 25.10.2023. — DOI: 10.48550/arXiv.2103.10385. — arXiv: 2103.10385 [cs]. — URL: http://arxiv.org/abs/2103.10385 (дата обр. 05.03.2024). — Пред. пуб.

16. LoRA: Low-Rank Adaptation of Large Language Models / E. J. Hu [и др.]. — 16.10.2021. — DOI: 10.48550/arXiv.2106.09685. — arXiv: 2106.09685 [cs]. — URL: http://arxiv.org/abs/2106.09685 (дата обр. 22.11.2023). — Пред. пуб.

17. *Zhang M.-L.*, *Zhou Z.-H.* ML-KNN: A Lazy Learning Approach to Multi-Label Learning // Pattern Recognition. — 2007. — 1 июля. — Т. 40, № 7. — С. 2038—2048. — ISSN 0031-3203. — DOI: 10.1016/j.patcog.2006.12.019. — URL: https://www.sciencedirect.com/science/article/pii/S0031320307000027 (дата обр. 04.04.2025).

18. Discriminative Nearest Neighbor Few-Shot Intent Detection by Transferring Natural Language Inference / J.-G. Zhang [и др.]. — 25.10.2020. — DOI: 10.48550/arXiv.2010.13009. — arXiv: 2010.13009 [cs]. — URL: http://arxiv.org/abs/2010.13009 (дата обр. 04.04.2025). — Пред. пуб.

19. CatBoost: Unbiased Boosting with Categorical Features / L. Prokhorenkova [и др.] // Advances in Neural Information Processing Systems. Т. 31 / под ред. S. Bengio [и др.]. — Curran Associates, Inc., 2018. — URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/14491b756b3a51daac41c24863285549-Paper.pdf.

20. The Faiss Library / M. Douze [и др.]. — 11.02.2025. — DOI: 10.48550/arXiv.2401.08281. — arXiv: 2401.08281 [cs]. — URL: http://arxiv.org/abs/2401.08281 (дата обр. 04.04.2025). — Пред. пуб.

21. *Malkov Y. A.*, *Yashunin D. A.* Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. — 14.08.2018. — DOI: 10.48550/arXiv.1603.09320. — arXiv: 1603.09320 [cs]. — URL: http://arxiv.org/abs/1603.09320 (дата обр. 04.04.2025). — Пред. пуб.

22. *Li Z.*, *Larson S.*, *Leach K.* Generating Hard-Negative Out-of-Scope Data with ChatGPT for Intent Classification. — 08.03.2024. — DOI: 10.48550/arXiv.2403.05640. — arXiv: 2403.05640 [cs]. — URL: http://arxiv.org/abs/2403.05640 (дата обр. 18.05.2025). — Пред. пуб.

23. *Hu R.*, *Khosmood F.*, *Edalat A.* Exploring Description-Augmented Dataless Intent Classification. — 25.07.2024. — DOI: 10.48550/arXiv.2407.17862. — arXiv: 2407.17862 [cs]. — URL: http://arxiv.org/abs/2407.17862 (дата обр. 04.04.2025). — Пред. пуб.

24. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines / O. Khattab [и др.]. — 05.10.2023. — DOI: 10.48550/arXiv.2310.03714. — arXiv: 2310.03714 [cs]. — URL: http://arxiv.org/abs/2310.03714 (дата обр. 04.04.2025). — Пред. пуб.

25. Optuna: A Next-generation Hyperparameter Optimization Framework / T. Akiba [и др.]. — 25.07.2019. — DOI: 10.48550/arXiv.1907.10902. — arXiv: 1907.10902 [cs]. — URL: http://arxiv.org/abs/1907.10902 (дата обр. 04.04.2025). — Пред. пуб.