ПРОГРАММНЫЙ АГЕНТ ОРГАНИЗАЦИИ ВЫЧИСЛЕНИЙ В КРУПНОМАСШТАБНЫХ СИСТЕМАХ УПРАВЛЕНИЯ[1]

Писарева О.М., Стефановский Д.В.
Государственный университет управления, Москва, Россия
om_pisareva@guu.ru, dv_stefanovskiy@guu.ru

Аннотация. Рассмотрены требования к программным агентам организации распределенных вычислений в крупномасштабных системах управления на базе MLOps. Представлены подходы и инструменты MLOps при внедрении моделей машинного обучения в реальные производственные процессы. Предложен эталонный типовой программный агент организации сетевой многоагентной системы распределенных вычислений.

Ключевые слова: программный агент, машинное обучение, MLOps, ML-модель, микросервис.

Введение

При попытках внедрения моделей машинного обучения (ML) в промышленную эксплуатацию пользователь зачастую сталкивается с многочисленными труднопреодолимыми проблемами. Именно поэтому не слишком много ML-моделей удается довести до промышленной эксплуатации. В частности, среди такого рода преград, тормозящих процесс внедрения ML-моделей, следует выделить: сложность предварительной обработки данных; эвристический характер создания и повторного обучения моделей; множественность методов оценки качества моделей, зачастую выдающих противоречивые результаты; нетривиальность процедур развертывания моделей и мониторинга их функционирования, а также множество других неприятностей, связанных со спецификой моделей искусственного интеллекта. В следствии этого в настоящее время чрезвычайно востребованы новые концептуальные подходы, методология и инструментарий внедрения и использования моделей машинного обучения в реальные производственные процессы. В работе сделана попытка представить новый инструментарий, поддержки жизненного цикла моделей машинного обучения на базе MLOps, что обеспечивает четкое направление и фокусировку на интересах организации для специалистов по изучению данных с измеримыми показателями.

1. Обзор

Создание приложений, использующих различные модели машинного обучения, предполагает разработку сложного программного обеспечения и способов их практического применения. Причем приемы, которые возможно использовать только в рамках исследования и разработки, по мере нарастания масштабности проекта могут стать либо недопустимы, либо потребуют сложных инфраструктурных и организационных решений. Все это отнимает много времени, сил и средств, особенно у разработчиков, ведущих несколько проектов. Кроме того, ручное внедрение инфраструктуры может привести к нюансам, которые могут фундаментально повлиять на операции и проекты по разработке моделей машинного обучения; например, модели машинного обучения могут использовать методы, которые нецелесообразно применять на практике.

Во многом поэтому становятся популярными подходы, методология и инструментарий, объединенный аббревиатурой MLOps, что является на сегодняшний день относительно новым направлением исследований и разработок, которому пока посвящено не так много научных работ, некоторые из которых хотелось бы упомянуть.

В частности, актуальность направления MLOps в области науки о данных описывается в работе [1], где авторы отмечают важность качества данных для системы MLOps и показывают, что его обеспечение и поддержка необходима на различных этапах развития машинного обучения. В работе [2] были проанализированы разнообразные аспекты качества данных, которые должны учитываться на различных этапах разработки и внедрения моделей машинного обучения. Выполняя совместный анализ влияния известных измерений качества данных и процесса машинного обучения, авторами также было показано то, как различные компоненты типичного конвейера MLOps могут быть спроектированы и описаны.

[1]*Статья выполнена при финансовой поддержке Министерства науки и высшего образования Российской Федерации, Соглашение № 075-15-2024-542*

Мониторинг и соответствующие ему проблемы обсуждались в работе Ризи и соавтора [3] на примере ряда примеров, готовых к производству решений с использованием инструментов с открытым исходным кодом. Наконец, в работе Уэйна [4] представлены современные тенденции и вызовы, связанные с применением MLOPS, причем особое внимание уделяется проблематике устойчивости, а также трактовке процессов и результатов моделирования. В статье [5] авторами представлены классы зрелости среды реализации сложных моделирующих комплексов и систем. Эта «модель зрелости» позволяет оценить уровень автономности существующих цифровых ML-инструментов, назначение которых - упрощение процесса внедрения ML-приложений производственной среде. Она основана на определении уровня автоматизации управления и конкретизирована в рамках Crisp-DM-методологии [6], [7]. Модель зрелости предполагает разделение прикладных решений на следующие уровни, характеризуемые наличием или отсутствием существенных составляющих процессов жизненного цикла создания и внедрения ML-моделей:
Уровень 0: DevOps не используется;
Уровень 1: MLOps не используется;
Уровень 2: доставка кода есть, но MLOps не задействован;
Уровень 3: автоматизированное обучение (AutoML);
Уровень 4: автоматизированное развертывание моделей;
Уровень 5: полная автоматизация операций MLOps.

Таким образом, можно сделать вывод, что MLOps представляет собой процесс, который похож на Devops, но учитывает особенности этапов жизненного цикла моделей машинного обучения.

При этом основной трудностью при формировании процессов в MLOps является развертывание подготовленной модели в производственных процессах, а также подходы связанные с определением качества данных, с которым работает модель и результатов ее применения на всем жизненном цикле. При наличии готовой модели ее дальнейшее использование на практике требует:

- *Инструментария автоматизированного обучения.* Специалисты по исследованию данных могут использовать множество различных фреймворков, языков и инструментов моделирования для создания уникальных моделей, для оценки качества модели необходимо не только знать показатели точности, но и возможность сравнить ее с другой моделью. Инструментарий MLOps должен позволять командам ИТ-операторов формировать модели в автоматизированном режиме(AutoML).
- *Инструментария упрощенного развертывания***.** Специалисты по исследованию данных могут использовать множество различных фреймворков, языков и инструментов моделирования, что может усложнить процесс развертывания. Инструментарий MLOps должен позволять командам ИТ-операторов на практике в средах как можно быстрей развертывать модели из различных фреймворков и языков.
- *Систем мониторинга процессов машинного обучения.* Традиционные программные инструменты мониторинга не подходят для мониторинга процесса разработки моделей машинного обучения. В отличие от них, мониторинг, который обеспечивает MLOps, предназначен для поддержки полного жизненного цикла модели, обеспечивая специфические для модели метрики, обнаружение изменения качества данных и результатов, а также другие немаловажные функции.
- *Систем управления жизненным циклом.* Развертывание — это лишь первый шаг в длительном жизненном цикле обновлений, чтобы поддерживать работоспособность ML-модели, разработчики и внедренцы должны регулярно тестировать модель и ее обновления перед практическим применением.
- *Инструментария обеспечения соответствия требованиям.* MLOps предлагает возможность отслеживания, контроля доступа и аудита, с целью минимизации рисков, предотвращения нежелательных изменений и обеспечения соответствия нормативным требованиям.

2. Организация распределенного конвейера MLOps

Традиционный подход создания ML-модели обычно направлен на её формирование на основе заранее подготовленных данных. Уделяя большое внимание экспериментам и проверке результатов работы модели с точки зрения точности, разработчики при этом мало анализируют ее готовность к использованию на практике, т.е. ее операциональность. При этом рабочий процесс моделирования обычно носит линейный и «ручной» характер, ему не хватает автоматизации и масштабируемости, необходимых для промышленной среды разработки и эксплуатации.

В целом порядок формирования модели соответствует Crisp-DM-методологии [6], [7]. а основным инструментом в рамках традиционного подхода для него является программное обеспечение на основе

Jupyter. Однако при этом не используется контроль версий данных или функций, что затрудняет сравнение результатов, отсутствует систематическое отслеживание версий модели, показателей производительности и метаданных эксперимента. Результаты прогнозирования, а также все артефакты модели сохраняются вручную в сервисе хранения данных, например, в облачном хранилище данных, что также затрудняет управление версиями моделей и данных для них.

Таким образом, возникает необходимость управлять большим количеством ациклических графов операторов (DAG), которые используют различные языки и библиотеки программ. В этой связи требуется простой инструмент управления небольшой группой сервисов, занимающихся интенсивными вычислениями. Такой инструмент представляет собой программный агент, который должен быть компактным, просто устанавливаться и иметь базовый набор для оркестрации вычислений на узле. С другой стороны, он должен иметь API интерфейс для возможности подключения к единой системе управления с дашборда мониторинга процесса. Принципиальная схема использования подобного программного агента представлена на рис 1.

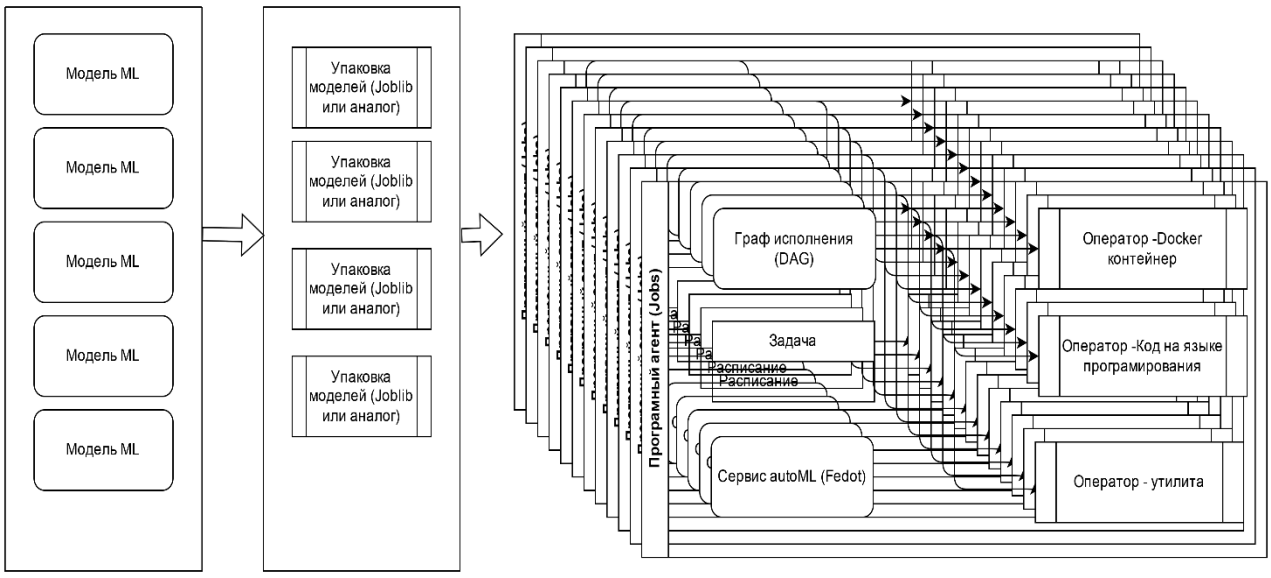

Рис. 1. Схема использования программного агента

Программный агент представляет собой открытую систему, помещенную в некоторую среду исполнения (узел кластера), при этом он обладает собственным поведением, удовлетворяющим некоторым экстремальным принципам [8]. Агент считается способным воспринимать информацию из внешней среды с ограниченным разрешением, обрабатывать ее на основе собственных ресурсов, взаимодействовать с другими агентами и воздействовать на среду в течение некоторого времени, преследуя свои собственные цели. Таким образом, агент – это программно или аппаратно реализованная система, обладающая следующими свойствами [9]:

- *автономность* – способность функционировать без прямого вмешательства людей или компьютерных средств, осуществляя при этом самоконтроль над своими действиями и внутренними состояниями [10];
- *общественное поведение (socialability)* - способность взаимодействия с другими агентами (а возможно, людьми) путем обмена сообщениями с помощью языков коммуникации;
- *реактивность* – способность воспринимать состояние среды (например, физического мира, пользователя – через пользовательский интерфейс, совокупности других агентов, сети Internet или сразу все этих внешних компонентов);
- *целенаправленная активность (pro-activity)* – способность агентов не просто реагировать на стимулы, поступающие из среды, но и осуществлять целенаправленное поведение, проявляя инициативу.

Эти свойства являются основным требованием, предъявляемым к программному агенту.

Одним из самых простых способов подключения программного агента является запуск скомпилированного программного кода. Обычно программный агент имеет сложную структуру, которая включает универсальный файл запуска и конфигурационные файлы, создание и интерпретация которых становится таким же сложным, что и программирование. Поэтому конфигурацию можно

определить внутри программного кода агента на его языке программирования. Это позволит скомпилировать программный агент в двоичный код и отправить его на узлы для организации вычислений, используя CI/CD подход. Это может принести различные преимущества, включая упрощенное тестирование.

Запуск программного агента осуществляется с использованием синтаксиса cron. Программный агент состоит из «Задач», которые используют заранее определенные операторы. Для «Задачи» задан граф исполнения.

Представим компоненты программного агента:

«Задача» – набор операторов, проводящих обработку данных (пример - выполнение заранее сконфигурированного докера или программы на языке Python);

«Оператор» – элемент программного агента, запускающий задачу определенного типа (пример - запуск докера, http-запрос или другая заранее определенная пользователем процедура общего вида);

«Граф исполнения» - инструкция, определяющая последовательно-параллельное выполнение задач операторами. Граф исполнения задает условия выполнения «Задач» следующим образом: задача автоматически запускается, если успешно выполнены все предыдущие задачи. Если для задачи не указаны предшественники, то она выполняется сразу. Таким образом, несколько задач не имеющих предшественников, выполняются параллельно. Программный агент имеет встроенный механизм потоковой передачи статуса задач на панель управления в режиме реального времени.

Операторы, которые используют задачи делятся на базовые, т.е. встроенные по умолчанию в программный агент, и пользовательские, т.е. те, которые определяет пользователь на языке программного агента. К базовым операторам целесообразно отнести:

- Command - выполняет команду оболочки;
- Get - выполняет запрос GET;
- Post - выполняет POST-запрос;
- Dockers-оператор, запускающий докера.

Приведем пример описания графа исполнения и задач в одном программном агенте на языке go:

```
package main

import (
	"log"
	"os"
	"stendml/goflow"
)

var options goflow.Options

func main() {
	dir, err := os.Getwd()
	if err != nil {
		log.Fatal(err)
	}
	options = goflow.Options{
		UIPath:      "ui/",
		Streaming:   true,
		ShowExamples: false,
		Public_dir:  dir,
	}
	gf := goflow.New(options)
	gf.AddJob(JobDockerBuild) // Граф исполнения 1
	gf.AddJob(JobDockerRun) // Граф исполнения 2
	gf.AddJob(JobCommands) // Граф исполнения 3
	gf.AddJob(JobGets) // Граф исполнения 4
	gf.Use(goflow.DefaultLogger())
	gf.Run(":8181")
}
```

Предложенный выше программный агент может быть скомпилирован и размещен в отдельном физическом или виртуальном узле для выполнения, что позволяет обеспечить его *автономность*. Для

реализации требования о *социальном подведении* агент должен реализовывать следующий минимальный набор конечных точек:

GET /api/health: проверьте работоспособность службы.

GET /api/jobs: список зарегистрированных заданий.

GET /api/jobs/{jobname}: получить подробную информацию о задании.

GET /api/jobruns: запрос и список запусков заданий.

POST /api/jobs/{jobname}/submit: отправить задание на выполнение.

POST /api/jobs/{jobname}/toggle: включение или выключение расписания заданий.

/stream: конечная точка возвращает события, отправленные сервером, с полезной нагрузкой data, соответствующей той, которая возвращается /api/jobruns.

В составе базовых операторов присутствуют операторы GET и POST, позволяющие обеспечивать свойство *реактивности*, то есть зависимость исполнения от внешних условий.

Ациклический граф исполнения задает целенаправленную активность и дает возможность исполнять альтернативные ветки команд, имитируя тем самым инициативу, связанную с выбором использования альтернативных моделей машинного обучения, в зависимости от данных, которые вместе с моделями, являются основными объектами управления MLOps.

Таким образом, вся совокупность графов исполнения объединяется в отдельный гиперграф исполнения, который позволяет объединять программные агенты, написанные на разных языках программирования, в том числе и python, а также заранее подготовленные контейнеры (Docker). При этом расписание программного агента может подчинятся расписанию физического узла, на котором развернут программный агент.

Гиперграф исполнения может быть разделен на домены, в которых управление и распределение ресурсов определяется администратором домена. Это снижает сложность управления гиперграфами вычислений, делает их независимыми от единой точки управления. Последнее позволяет регламентировать горизонтальное масштабирование и выделять ресурсы конкретным графам исполнений. Домены графов исполнения удобно организовывать по типу моделей, тогда формируются комплексные единицы ресурсов, которые состоят из ядер, в том числе и специфических, оперативной памяти и долгосрочного хранилища данных. Это позволяет обеспечить учет и контроль полезности и востребованности использования вычислительных ресурсов, а также является основой для их экономической оценки.

3. Заключение

Выполнен обзор подходов, связанных с применением MLOps, а также представлены модели зрелости. Предложен подход к разработке многоагентных систем, позволяющий масштабировать программные агенты объединенные в гиперграф исполнения в виде набора микросервисов, обеспечивающий возможность добавлять новые агенты в гиперграф исполнения, а затем анализировать технические параметры, используя теорию графов. Гиперграф позволяет регламентировать доступ к вычислительным ресурсам, определять их полезность и проводить экономическую оценку используемых ресурсов, не только исходя из их стоимости и востребованности, что позволяет улучшить существующие практики развития центров обработки данных.

Литература

1. *Renggli C. [и др.].* A Data Quality-Driven View of MLOps / /IEEE Data Engineering Bulletin March. – 2021.
2. *Mboweni T., Masombuka T., Dongmo C.* A systematic review of machine learning devops //2022 international conference on electrical, computer and energy technologies (ICECET). – IEEE, 2022. – С. 1-6.
3. *Rizzi W. [и др.].* Explainability in predictive process monitoring: When understanding helps improving //International Conference on Business Process Management. – Cham: Springer International Publishing, 2020. – С. 141-158.
4. Weyns *D. [и др.].* Towards a Research Agenda for Understanding and Managing Uncertainty in Self-Adaptive Systems //ACM SIGSOFT Software Engineering Notes. – 2023. – Т. 48. – №. 4. – С. 20-36.
5. *Mannan M. A., [и др.].* A Maturity Level Framework for Practicing Machine Learning Operations in CI/CD For Software Deployment //Journal of Independent Studies and Research Computing. – 2024. – Т. 22. – №. 1. – С. 47-53.
6. *Schröer C., Kruse F., Gómez J.M.* A systematic literature review on applying CRISP-DM process model //Procedia Computer Science. – 2021. – Т. 181. – С. 526-534.
7. *Савилова Е.А.* Обзор методологий управления проектами в области искусственного интеллекта // Вестник науки и образования. 2021. №6-1 (109). С. 29-33. URL: https://cyberleninka.ru/article/n/obzor-metodologiy-upravleniya-proektami-v-oblasti-iskusstvennogo-intellekta (дата обращения: 03.06.2024).

8. *Тарасов В.Б.* От многоагентных систем к интеллектуальным организациям: философия, психология, информатика. – М.: УРСС, 2002. – 352 с.
9. *Жмурко С.А.* Обобщенная модель агента и многоагентной системы // Известия ЮФУ. Технические науки. 2008. №4. С. 115-120. URL: https://cyberleninka.ru/article/n/obobschennaya-model-agenta-i-mnogoagentnoy-sistemy (дата обращения: 03.06.2024).
10. *Трахтенгерц Э.А.* Компьютерная поддержка принятия решений. – М.: Синтег,1998. – 376 с.