Постановка задачи для RL-исследователя: Динамическое управление инвентарем в авиационной отрасли с помощью Reinforcement Learning

1. Проблема и Цель

Мы решаем классическую проблему динамического управления доходами (Dynamic Revenue Management) в контексте пассажирских авиаперевозок. Основная цель — разработка политики управления вместимостью самолета, которая максимизирует совокупную выручку от одного рейса за весь период продаж. Система должна в реальном времени принимать решения о доступности различных тарифных классов для бронирования.

2. Формализация в рамках Марковского процесса принятия решений (MDP)

- **Среда (Environment):** Изолированный рейс (flight leg) с фиксированной емкостью C и набором из K вложенных тарифных классов (fare classes), F = {f_1, ..., f_K}.
- **Эпизод:** Период продаж для одного рейса, дискретизированный по времени. Длительность эпизода T (например, 365 дней).
- **Временной шаг (Timestep t):** Один календарный день до вылета. t изменяется от T до 1.
- **Пространство состояний (State Space, S):**
  Состояние s_t ∈ S в момент времени t представляет собой вектор, включающий:
  - **Временной фактор:** t — количество дней до вылета.
  - **Состояние инвентаря:** b*t — вектор размерности K, где b*{t,k} — количество уже проданных мест в классе f_k и выше.
  - **Конкурентная среда:** p_t — вектор, представляющий наблюдаемые тарифы конкурентов.
  - **Прогностическая информация:** d_t — вектор прогнозируемого остаточного спроса для каждого класса f_k.
- **Пространство действий (Action Space, A):**
  Действие a*t ∈A в момент t — это определение уровней авторизации (authorization levels) для каждого тарифного класса. Это вектор a_t размерности K, где a*{t,k} — это максимальное количество мест, которое может быть продано в классе f*k и ниже. Учитывая вложенную структуру (nesting), если класс f_k открыт, то все более дорогие классы f*{k+1}, ..., f_K также открыты.
- **Функция вознаграждения (Reward Function, R):**
  Вознаграждение r*t = R(s_t, a_t) — это выручка, полученная за интервал времени (t, t-1]. Целью является максимизация недисконтированного совокупного вознаграждения за эпизод: G = Σ{t=1}^{T} r_t.

3. Ключевые исследовательские вызовы

1. **Высокая размерность пространства состояний и действий (Curse of Dimensionality):** Задача требует эффективных методов аппроксимации функций ценности (value functions) и политик (policies) с помощью глубоких нейронных сетей.
2. **Нестационарность среды:** Распределение спроса и стратегии конкурентов не являются стационарными, что требует от агента способности к быстрой адаптации. Необходимо исследовать методы, устойчивые к дрейфу данных и изменению динамики среды.
3. **Длинный горизонт планирования и отложенное вознаграждение:** Эффект от действия, принятого на раннем этапе продаж, проявляется в конце эпизода. Это создает классическую проблему кредитного назначения (credit assignment).
4. **Безопасное и эффективное исследование (Safe & Efficient Exploration):** Обучение в симулированной среде требует баланса между исследованием потенциально рискованных, но высокодоходных стратегий и эксплуатацией уже известных. При переходе к реальному миру исследование должно быть строго ограничено для минимизации рисков.
5. **Разрыв между симуляцией и реальностью (Sim-to-Real Gap):** Требуется разработка методов валидации симулятора и техник обучения, обеспечивающих робастность и успешный перенос политики из симулированной среды в реальную операционную деятельность.

Наша задача — провести исследовательскую работу по проектированию, обучению и оценке RL-агента, способного эффективно решать данную задачу, с фокусом на вышеупомянутых вызовах.

1. Model-Free подход

Философия: "Мир слишком сложен, чтобы его моделировать. Давайте не будем тратить время на построение карты, а просто научимся идеально ориентироваться на местности методом проб и ошибок".

- **Фокус:** На прямом обучении связки "состояние → действие" (Policy-based методы, как PPO) или "состояние-действие → ценность" (Value-based методы, как Q-learning/SAC). Не пытаемся понять почему мир так устроен.
- **Методы:** Policy Gradients, Actor-Critic (A2C, A3C), PPO, SAC, DDPG. Это проверенные методы современного RL.
- **Сильные стороны:**
    - **Универсальность:** Могут работать даже тогда, когда физика мира неизвестна или слишком сложна для моделирования.
    - **Асимптотическая производительность:** При достаточном количестве тренировок могут достичь сверхчеловеческих результатов.
- **Слабые стороны:**

    - **Крайняя неэффективность по данным (Sample Inefficiency):** Требуют миллионов, а иногда и миллиардов, взаимодействий со средой, чтобы научиться чему-то стоящему.
- **Для нас:** Наш первоначальный план. Это надежный, проверенный, но очень требовательный к симулятору путь.

Практическая реализация:

1. **Центральный элемент:** Высокопроизводительный, но детерминированный **симулятор**. Это наша главная инвестиция. Он должен быть написан на компилируемом языке (C++, Rust) или с использованием высокооптимизированных Python-библиотек (JAX, Polars, Numba) для максимальной скорости.
2. **Процесс обучения:**
    - Мы запускаем **массивное распределенное обучение** с помощью фреймворка типа **Ray**. Наш Kubernetes-кластер будет состоять из сотен CPU-ядер.
    - На каждом ядре будет работать свой экземпляр симулятора ("worker"), который генерирует опыт.
    - Центральный "learner" на GPU будет непрерывно собирать этот опыт и обновлять веса одной нейросети (наш агент, скорее всего, на базе **PPO**).
3. **Архитектура агента:** Мы можем позволить себе сложную архитектуру, например, **Трансформер**, потому что у нас будет практически неограниченный объем данных из симулятора для его обучения.
4. **Что дает нам Трансформер (и почему это прорыв):** Архитектура Трансформера изначально была создана для обработки последовательностей (текста). Мы можем представить историю бронирования рейса как "предложение", где каждый день — это "слово".

- **Как это работает:** На вход Трансформеру мы подаем не один state_vector, а **последовательность векторов** за последние, скажем, 14 дней: [state_t-13, state_t-12, ..., state_t].
- **Механизм внимания (Self-Attention):** Это сила Трансформера. Он может "обратить внимание" на разные моменты в прошлом. Например, принимая решение сегодня, он может увидеть, что 7 дней назад был резкий скачок спроса, и придать этому событию больший вес, чем вчерашнему затишью. Он учится находить **причинно-следственные связи во времени**.
- **Что он сможет уловить:**
    - **Тренды:** "За последнюю неделю загрузка стабильно росла на 2% в день".
    - **Ускорение/замедление:** "Темп бронирования резко замедлился после того, как конкурент снизил цены 3 дня назад".
    - **Сезонные паттерны:** "Ага, сейчас ровно за 30 дней до вылета, в это время обычно начинается волна покупок от бизнес-пассажиров".

4. **Результат:** Мы получим агента, который идеально "заточен" под наш симулятор. Он будет демонстрировать сверхчеловеческую производительность внутри этой симулированной реальности.
5. **Главный риск и как мы с ним боремся:** Риск — **"разрыв между симуляцией и реальностью" (Sim-to-Real Gap)**. Если наш симулятор неточен, агент будет бесполезен. Поэтому параллельно мы постоянно работаем над валидацией и улучшением симулятора, сравнивая его поведение с реальными данными.
6. **Гипотеза (Робастность через разнообразие):** Агент, обученный в среде, которая генерирует множество разнообразных стохастических сценариев, будет значительно более устойчив к рыночным шокам, чем агент, обученный на одном детерминированном симуляторе.
    - **Исследование:** Разработать генеративную модель среды (начиная с простых вероятностных методов и двигаясь к GAN/диффузионным моделям). Сравнить робастность агентов, обученных в детерминированной и генеративной средах, на наборе стресс-тестов.
    - **Строим Генеративную Модель Спроса:** Мы используем современные генеративные модели (например, **диффузионные модели** или **Generative Adversarial Networks - GANs**) для моделирования не просто среднего спроса, а всего **распределения вероятностей** будущих потоков бронирований. Мы обучаем эту модель на всех исторических данных, чтобы она "поняла" структуру, сезонность и случайность реального спроса.
    -
    - **Меняем парадигму обучения RL-агента: Обучение превращается в управление рисками.** Теперь задача агента — не просто максимизировать выручку в одном сценарии. Его функция вознаграждения может быть изменена, чтобы учитывать риски. Например, мы можем использовать метрики вроде **CVaR (Conditional Value at Risk)**. Агент будет стараться максимизировать выручку, но при этом "отсекать" самые плохие исходы, делая его стратегию более безопасной.
    - **Агент учится адаптивности.** Принимая решение сегодня, он знает, что завтра мир может измениться. Это заставляет его принимать более гибкие решения, которые оставляют "пространство для маневра" (например, не закрывать все дешевые классы слишком рано, на случай если спрос окажется ниже ожидаемого).
    - **Метрика успеха:** Агент из генеративной среды показывает значительно меньшее падение производительности в стрессовых сценариях ("черные лебеди").

2. Model-Based подход

Философия: "Действовать вслепую — глупо. Прежде чем сделать шаг, нужно построить модель мира и продумать последствия на несколько ходов вперед. Интеллект — это способность планировать".

- **На чем сфокусированы:** На двух задачах. Сначала — научиться **модели мира** (World Model), которая может предсказывать, что произойдет в следующий момент времени, если совершить определенное действие. Затем — использовать эту модель для "воображаемого" планирования или обучения агента.
- **Методы:** AlphaGo/AlphaZero (идеальный пример!), Dreamer, MuZero, планирование на основе дерева поиска (MCTS).
- **Сильные стороны:**
  - **Высокая эффективность по данным:** Требуют на порядки меньше реальных взаимодействий, так как большую часть "опыта" агент получает, "мечтая" или "планируя" внутри своей модели мира.
  - **Интерпретируемость:** Можно заглянуть в модель мира и понять, как агент "видит" будущее.
- **Слабые стороны:**
  - **Ошибка моделирования:** Если модель мира неточна ("мусор на входе"), то все планы, построенные на ее основе, будут бесполезны ("мусор на выходе"). Агент может начать эксплуатировать ошибки в своей модели.
- **Для нас:** Наш подход с "генеративной средой" — это шаг в эту сторону. Мы пытаемся построить более совершенную модель нашего мира.

Практическая реализация:

1. **Центральный элемент: Модель Мира (World Model)**. Это нейросеть (вероятно, комбинация RNN и Mixture Density Networks), которая учится отвечать на вопрос: "Если сейчас состояние S и мы совершаем действие A, каким будет следующее состояние S' и какое вознаграждение R мы получим?". Она учится предсказывать динамику загрузки, цены конкурентов и спрос.
2. **Процесс обучения:**
   - **Шаг А: Обучение Модели Мира.** Мы "кормим" ее всеми нашими историческими данными, чтобы она научилась предсказывать следующий шаг. Это supervised learning, что намного стабильнее, чем RL.
   - **Шаг Б: Обучение агента "в воображении".** Как только Модель Мира готова, мы обучаем нашего RL-агента (например, на базе **SAC**) полностью внутри нее. Агент "мечтает" о будущих рейсах, а Модель Мира рассказывает ему, что произойдет. Это происходит невероятно быстро, так как не требует запуска тяжелого симулятора.
3. **Архитектура агента:** Может быть относительно простой (даже MLP), так как вся сложность "спрятана" в Модели Мира.

4.
    5.
        - **Гипотеза Д.1 (Интеллектуальное Состояние):** Большую часть прироста производительности можно получить не за счет усложнения архитектуры агента, а за счет обогащения вектора состояний производными признаками (динамика, ускорение, сравнение с историей).
        - **Исследование:** Провести масштабную работу по Feature Engineering. Создать "идеальный" вектор состояния и протестировать его с самой простой моделью (A.1).
        - **Метрика успеха:** Агент A.1 на обогащенных данных показывает результаты, сопоставимые или превосходящие более сложные модели (A.2, A.3) на "сырых" данных.

- **Гипотеза Д.2 (Декомпозиция):** Единый "монолитный" агент менее эффективен, чем команда узкоспециализированных агентов, каждый из которых обучен на своем сегменте данных (например, "бизнес-маршруты", "туристические", "ближнемагистральные").
    - **Исследование:** Разделить данные на сегменты. Обучить несколько простых агентов и сравнить их суммарную производительность с одним большим агентом.
    - **Метрика успеха:** "Команда специалистов" показывает более высокую общую выручку и лучшую стабильность на своих сегментах.

4. **Результат:** Мы получаем агента, который очень эффективно использует данные и может планировать свои действия на несколько шагов вперед. Он может "разыгрывать" в уме целые сценарии: "А что, если я сейчас закрою этот класс? Модель Мира предсказывает, что тогда конкурент поднимет цену, и я смогу продать более дорогие билеты".
5. **Главный риск и как мы с ним боремся: Эксплуатация ошибок модели.** Агент может найти "баг" в Модели Мира (например, сценарий, где она ошибочно предсказывает гигантскую выручку) и начать его безжалостно эксплуатировать. Мы боремся с этим, добавляя в обучение "шум" и постоянно перепроверяя предсказания Модели Мира на реальных данных.

3. Offline RL - подход

Философия: "Мир полон данных о чужих успехах и провалах. Зачем рисковать и совершать свои ошибки, если можно научиться на огромном архиве уже существующих данных? Безопасность и использование того, что есть — превыше всего".

- **На чем сфокусированы:** На разработке алгоритмов, которые могут извлекать оптимальную стратегию из статичного набора данных **без какого-либо нового взаимодействия со средой**.
- **Методы:** CQL (Conservative Q-Learning), IQL, Decision Transformer.
- **Сильные стороны:**
  - **Безопасность:** Идеально для задач, где цена ошибки в реальном мире высока (медицина, робототехника, финансы).
  - **Экономия ресурсов:** Позволяет использовать уже накопленные гигантские датасеты.
- **Слабые стороны:**
  - **Ограниченность данных:** Агент никогда не сможет стать лучше, чем лучшая стратегия, которая хоть как-то представлена в данных. Он не может исследовать что-то радикально новое.
- **Аналогия:** Обучение пилота на записях "черных ящиков" и логах тысяч успешных полетов. Он учится лучшим практикам, но не садится за штурвал, чтобы попробовать что-то новое.
- **Для нас:** Наш гибридный план, начинающийся с Offline RL, напрямую заимствует философию этой школы как самый быстрый и безопасный первый шаг.

Практическая реализация:

1. **Центральный элемент:** Огромный, тщательно очищенный **исторический датасет**. Все наши усилия направлены на инжиниринг данных. Мы создаем таблицу вида (state, action, reward, next_state) за последние 3-5 лет.
2. **Проектируем новый state_vector:**
   - **Базовые признаки (как и раньше):** days_to_departure, load_factor_total, competitor_prices_min...
   - **Признаки Динамики (первого порядка):** Мы вычисляем дельты за разные периоды.
     - load_factor_delta_1d, load_factor_delta_7d, load_factor_delta_30d (изменение загрузки за день, неделю, месяц).
     - booking_rate_last_3d (средняя скорость бронирования за последние 3 дня).
     - competitor_price_change_last_24h (изменил ли конкурент цену за сутки).
   - **Признаки Динамики (второго порядка):** Мы вычисляем "ускорение".
     - booking_acceleration_7d (насколько текущая недельная скорость бронирования выше/ниже предыдущей).
     - Это позволяет модели понять: спрос не просто растет, а растет все быстрее.
   - **Контекстуальные и Циклические Признаки:** Мы помогаем модели понять время.
     - day_of_week, month_of_year (в виде синусно-косинусных преобразований, чтобы показать цикличность).
     - is_holiday_season, is_school_vacation.

        - days_since_last_price_change (как давно мы меняли цены).
    - **Сравнительные Признаки:** Мы сравниваем текущий рейс с "эталоном".
        - load_factor_vs_historical_avg (насколько текущая загрузка опережает/отстает от средней для этого рейса за 90 дней до вылета).
        - current_price_vs_initial_price.
3. **Процесс обучения:**
    - Никакого симулятора. Никакого взаимодействия. Мы берем алгоритм **Conservative Q-Learning (CQL)**.
    - Мы запускаем обучение на одной (или нескольких) мощных машинах с GPU. Процесс похож на обычное обучение нейросети в supervised learning.
    - Алгоритм CQL будет штрафовать агента за любые попытки оценить действия, которых не было в датасете. Это заставляет его быть консервативным и придерживаться уже проверенных стратегий.
4. **Архитектура агента:** Скорее всего, MLP на обогащенных признаках, так как нам нужно извлечь максимум сигнала из ограниченных данных.
5. **Почему MLP — это "хорошо, но недостаточно":** MLP (многослойная полносвязная сеть) отлично работает с табличными данными. Она смотрит на state_vector в момент времени t и принимает решение. Но она не видит динамику. Для нее состояние рейса за 90 дней до вылета и за 89 дней — это две независимые картинки. Она не улавливает тренды, ускорения, паттерны.
6. **Результат: Очень быстро (за несколько недель)** получаем первого агента. Он будет гарантированно не хуже текущей системы (так как учился на ее решениях) и, скорее всего, немного лучше, так как научился комбинировать лучшие из прошлых стратегий.
7. **Главный риск и как мы с ним боремся: Агент не может быть инновационным.** Он никогда не придумает радикально новую стратегию. Мы боремся с этим, используя этот подход как **первый шаг**. Получив этого "консервативного" агента, мы можем затем аккуратно дообучать его в симуляторе (fine-tuning), чтобы он научился исследовать новые возможности.

4. Подход "Универсалистов" (LLM Agent)

Философия: "Не нужно создавать тысячи разных узкоспециализированных "интеллектов". Нужно создать один, но очень большой и универсальный разум (Foundation Model), который уже обладает знаниями о мире, а затем научить его применять эти знания для решения конкретных задач".

- **На чем сфокусированы:** На адаптации гигантских, предобученных моделей (в основном, Трансформеров типа GPT) для решения задач RL. RL не как на задачу оптимизации, а как на задачу **предсказания последовательности** "правильных" действий.
- **Методы:** Decision Transformer, Gato (DeepMind), использование LLM с промптингом (ReAct, Chain-of-Thought).
- **Сильные стороны:**
    - **Встроенные знания о мире:** Могут использовать контекст, недоступный другим школам (например, новости).
    - **Few-shot адаптация:** Потенциально могут быстро адаптироваться к новым задачам с минимальным дообучением.
- **Слабые стороны:**
    - **Фундаментальные проблемы:** Отсутствие цели максимизации вознаграждения, проблема "заземления" (понимания чисел и физического мира), галлюцинации.
    - **Вычислительная стоимость:** Невероятно дороги в обучении и использовании.
- **Для нас:** Это рискованный, но потенциально революционный путь. Им мы завершим

Практическая реализация:

1. **Центральный элемент: Большая Языковая Модель (LLM)**, дообученная (fine-tuned) на наших данных. Это может быть модель вроде Llama 3 или Qwen или Deepseek.
2. **Процесс "обучения" (скорее, адаптации):**
    - **Шаг А: Создание датасета для Fine-tuning.** Мы превращаем наши структурированные данные в текст. Например: "Инструкция: Ты — эксперт по управлению доходами. Проанализируй ситуацию и предложи оптимальные уровни авторизации. Ситуация: Рейс Москва-Сочи, до вылета 45 дней, эконом-класс загружен на 60%, бизнес на 20%. Конкурент 'Аэрофлот' продает билеты по 8500 рублей. Погода в Сочи ожидается солнечная. Идеальный ответ: {'auth_levels': [1,1,1,0,0,....]}".
    - **Шаг Б: Дообучение LLM.** Мы дообучаем модель на этом датасете, чтобы она научилась говорить на языке нашей задачи и выдавать ответы в строгом JSON-формате.
3. **Интеграция в production:**
    - Наш сервис в реальном времени формирует текстовый промпт, описывающий текущую ситуацию на рейсе.
    - Он отправляет этот промпт на API нашей дообученной LLM.
    - Он получает в ответ JSON, парсит его и отправляет управляющие команды в систему бронирования.
4. **Результат:** Мы получаем систему, которая потенциально может учитывать самый широкий контекст (новости, погоду, события) и может "объяснить" свои решения на естественном языке.

5. **Главный риск и как мы с ним боремся: Галлюцинации и непредсказуемость.** Модель может выдать невалидный JSON, абсурдные цифры или просто "зависнуть". Мы боремся с этим, внедряя **жесткую систему валидации** на выходе (проверка формата, диапазона значений) и **систему "предохранителей"** (если LLM выдает чушь, система автоматически переключается на безопасную, консервативную стратегию).