**Текст выступления:**

**Слайд 2 (Введение)**  
«Веб-парсинг — ключ к автоматизации сбора данных из интернета. Однако сайты бывают разными: одни отдают простой HTML, другие — тяжёлые SPA. В классических подходах для каждого сайта пишут и поддерживают свой ручной скрипт-парсер. Это трудоёмко и неуниверсально. В проекте автоматизация этого процесса возложена на большие языковые модели.»

**Слайд 3 (Существующие решения и их недостатки)**  
«Как было сказано ранее, классические методы, что и статический парсинг, и динамический через Selenium требуют ручного написания и поддержки скриптов-парсеров для каждой структуры. А при изменении верстки веб страницы - парсер ломается, а универсального «one-size-fits-all» решения нет.»

**Слайд 4 (LLM-подход и семантический кэш)**  
«Следовательно, возникает вопрос автоматизации процесса парсинга и тут на помощь приходят большие языковые модели.  
В проекте реализовано два режима, два способа парсинга.  
1) В режиме Structuring модель извлекает ответ на запрос пользователя напрямую из очищенного текста страницы.   
2) В режиме Codegen используется две модели. Первая модель - HintGen генерирует подсказки, извлекает селекторы и дает few-shot примеры того, как необходимо ответить на запрос пользователя. А вторая модель CodeGen на их основе создаёт Python-скрипт-парсер и выполняет его.  
  
Однако, у такого подхода – есть недостатки: это дороговизна запросов в LLM и время инференса модели.   
Данная проблема была решена внедрением БД SQLite для хранения ссылок, запросов и закэшированного кода-парсера, а так же векторной БД ChromaDB для хранения embedding представления запросов. При повторных или семантически похожих запросах – не отправляется запрос в LLM, а достаётся готовый парсер из БД.

**Слайд 5 (Постановка задачи)**  
«Цель проекта - оптимизировать сбор данных из открытых веб-источников за счёт автоматизации написания парсеров веб-страниц с помощью больших языковых моделей (LLM). Для этого выделены шесть задач: определение типа страницы; две стратегии очистки HTML; внедрение режимов Structuring и Codegen; построение семантического кэша; разработка веб-интерфейсов; проведение экспериментального исследования.»

**Слайд 6 (Архитектура решения)**  
«На этой схеме показан рабочий процесс. По введённому URL и запросу детектор определяет необходимость JS-рендеринга. Затем HTML загружается через requests или Selenium и очищается. Диспетчер переключается между Structuring и Codegen. В последнем случае две модели LLM генерируют скрипт, который выполняется и сохраняется в кэше. После чего возвращается в виде JSON.»

**Слайд 7 (Экспериментальное исследование)**  
«В качестве тестирования системы был проведен эксперимент. Было выбрано несколько сайтов и несколько запросов для каждого. На данном слайде показана часть его результатов, с полной версией вы можете ознакомится в отчете работы.  
В таблице представлены замеры для трёх сайтов. При первом (cold) запросе Codegen занимал в среднем 27 секунд, при повторном (warm с сохраненном кэше) — 5–6 секунд; В режиме Structuring — 8–12 секунд. В тестах система верно обрабатывала все запросы и корректно извлекала данные.»

**Слайд 8 (Примеры JSON-ответов)**  
«Здесь несколько примеров ответов в формате JSON.»

**Слайд 9 (Выводы)**  
В итоге в проекте создана гибкая и расширяемая система, которая автоматически обрабатывает и парсит как статические, так и динамические страницы. Для генерации парсеров применяются два режима работы с LLM: прямое извлечение ответа из текста и Codegen с подсказками, селекторами и few-shot примерами. Чтобы ускорить повторные запросы, интегрирован семантический кэш, позволяющий повторно использовать уже сгенерированные скрипты. Система предоставляет несколько интерфейсов — от чисто библиотечного до REST API, веб-фронтенда и Gradio — что облегчает интеграцию. Экспериментальные замеры на реальных сайтах подтвердили надёжность и эффективность предложенного подхода

**Слайд 10 (QR-код)**  
« Воспользоваться проектом можно, отсканировав QR-код для перехода на Hugging Face Space.»