DOI https://doi.org/10.46845/2541-8254-2026-4(56)-77-77
УДК 004.42:004.89



РАЗРАБОТКА КРОССПЛАТФОРМЕННОГО ПОЛЬЗОВАТЕЛЬСКОГО БРАУЗЕРА С ИНТЕГРАЦИЕЙ ФУНКЦИОНАЛА ИИ-АНАЛИЗА СТРАНИЦ

А. К. Красовский, студент 3-го курса
E-mail: andrew.krasovsky39@gmail.com
ФГБОУ ВО «Калининградский государственный технический университет»

А. Н. Ильяшов, старший преподаватель
E-mail: andrej.ilyashov@klgtu.ru
ФГБОУ ВО «Калининградский государственный технический университет»

В работе представлена разработка кроссплатформенной браузерной оболочки Intra Browser на базе движка отображения Chromium со встроенным модулем локального искусственного интеллекта для анализа содержимого веб-страниц. Рассмотрены технические ограничения существующих облачных AI-инструментов, связанные с конфиденциальностью данных и операционными расходами. Предложена многоуровневая архитектура приложения на базе фреймворка Electron, объединяющего движок отображения Chromium и среду исполнения Node.js, в которой языковая модель запускается непосредственно на устройстве пользователя через библиотеку llama.cpp. Реализованы интегрированные в страницу операции резюмирования, перевода и объяснения программного кода, а также система управления сетевым доступом приложения с тремя режимами фильтрации трафика. Описаны механизмы межпроцессного взаимодействия, обработка контекста веб-страницы и принципы работы с моделями в формате GGUF. Приведены характеристики тестового стенда, результаты замеров производительности при различных уровнях квантования модели, а также сравнительная характеристика разработанного решения с известными аналогами.

Ключевые слова: *веб-браузер, браузерная оболочка, локальный искусственный интеллект, языковая модель, Electron, Chromium, llama.cpp, GGUF, кроссплатформенная разработка, конфиденциальность данных, корпоративная среда.*

ВВЕДЕНИЕ

За последние годы инструменты на основе искусственного интеллекта (ИИ) прочно вошли в практику работы с цифровым контентом: они применяются для перевода, реферирования, анализа программного кода и поддержки принятия решений [1, 2]. Однако подавляющее большинство таких сервисов реализовано в виде облачных API, что предполагает обязательную передачу пользовательских данных на серверы внешних поставщиков и постоянное подключение к сети.

Для организаций, в которых обработка информации регламентирована требованиями конфиденциальности, – высших учебных заведений, научно-исследовательских институтов, промышленных предприятий – облачные AI-сервисы оказываются неприменимыми по трем основным причинам. Во-первых, передача внутренних документов на серверы сторонних поставщиков нарушает регламенты обработки персональных данных и сведений ограниченного распространения [3]. Во-вторых, лицензирование таких сервисов из расчета на каждого сотрудника создает существенные операционные расходы, не всегда соответствующие бюдже-

там организаций. В-третьих, отсутствует контроль над тем, какие именно фрагменты данных и в каком объеме обрабатываются за пределами организации, что затрудняет аудит и затрудняет соответствие отраслевым стандартам безопасности.

Современные браузеры общего назначения (Google Chrome, Mozilla Firefox, Brave) не предоставляют встроенных средств локального ИИ-анализа: их функциональность ориентирована на отображение веб-контента, а интеллектуальные надстройки реализуются как сторонние расширения, обращающиеся к удаленным сервисам через сетевые API. Браузерные расширения принципиально ограничены в возможностях: они не могут запускать вычислительно емкие модели в едином процессе с движком отображения и не имеют доступа к настройкам сетевого стека приложения. Это формирует научно-практическую проблему: разработать программное средство, в котором функции ИИ-анализа интегрированы непосредственно в браузер на уровне приложения и выполняются на стороне клиента без обязательной передачи данных в сеть.

Актуальность работы определяется растущим спросом на инструменты обработки текста и кода в условиях, исключающих передачу данных третьим лицам. Научная новизна заключается в подходе, при котором браузер рассматривается не как «окно в Интернет», а как платформа для выполнения локальных интеллектуальных операций над содержимым веб-страниц, и в архитектурных решениях, обеспечивающих такую интеграцию.

ОБЪЕКТ ИССЛЕДОВАНИЯ

Объектом исследования является кроссплатформенное клиентское приложение класса «веб-браузер» с интегрированным модулем локального ИИ-анализа содержимого страниц. С архитектурной точки зрения разработанное приложение представляет собой браузерную оболочку (browser shell), использующую встроенный движок отображения Chromium и расширяющую его собственными модулями локального ИИ-анализа и управления сетевым доступом, в отличие от полнофункционального браузера с собственным движком отображения. Предметом исследования выступают методы и архитектурные решения интеграции языковых моделей в десктопное и мобильное приложение в одном процессе с движком отображения, а также механизмы управления сетевым доступом на уровне приложения для применения в корпоративных и образовательных средах.

ЦЕЛЬ И ЗАДАЧИ ИССЛЕДОВАНИЯ

Цель работы – разработать кроссплатформенный браузер, в котором ИИ-ассистент является встроенной частью системы, а не внешним облачным сервисом, и обеспечивает возможность анализа содержимого веб-страниц без обязательной передачи данных в сеть.

Для достижения поставленной цели сформулированы следующие задачи: 1) спроектировать и реализовать многовкладочный браузер с поддержкой приватного режима на базе движка Chromium; 2) интегрировать в приложение локальную языковую модель, способную работать без подключения к Интернету, обеспечив при этом приемлемое время отклика на типовой пользовательской конфигурации; 3) реализовать набор интеллектуальных операций над содержимым веб-страницы (резюмирование, перевод, объяснение фрагментов программного кода) с возможностью вызова из контекстного меню и боковой панели; 4) разработать систему управления сетевым доступом с тремя режимами – «Интернет», Smart и «Только интрасеть» – на основе перехвата HTTP- и HTTPS-запросов уровня приложения; 5) спроектировать модуль децентрализованного обмена данными между устройствами в локальной сети организации; 6) реализовать мобильную версию браузера для операционной системы Android.

МЕТОДЫ ИССЛЕДОВАНИЯ

В ходе исследования применялись методы системного анализа, объектно-ориентированного проектирования, кроссплатформенной разработки программного обеспечения, а также сравнительного анализа существующих решений. Для количественной оценки

производительности проводилось измерение времени отклика модели и потребления оперативной памяти на типовой пользовательской конфигурации.

Разработка прототипа осуществлялась на персональном компьютере с процессором Intel Core i7-12700H, 32 ГБ оперативной памяти DDR4-3200, видеокартой NVIDIA GeForce RTX 3060 (6 ГБ VRAM), твердотельным накопителем NVMe объемом 1 ТБ под управлением операционной системы Windows 11 (64-разрядная). Апробация работоспособности приложения и измерение характеристик производительности проводились дополнительно на типовой конфигурации офисного класса (Intel Core i5-1135G7, 8 ГБ оперативной памяти DDR4-3200, встроенная графика Intel Iris Xe) и на мобильном устройстве с процессором Qualcomm Snapdragon 8 Gen 1 (8 ГБ оперативной памяти, операционная система Android 14). Подобный выбор тестовых конфигураций позволил оценить применимость решения как на устройствах разработчика, так и на типовых рабочих местах сотрудников организации.

В качестве технологической основы выбран фреймворк Electron, объединяющий движок отображения Chromium и среду исполнения JavaScript Node.js [4]. Архитектура Electron разделяет приложение на главный процесс (main process), управляющий жизненным циклом окон и системными ресурсами, и процессы рендеринга (renderer process), выполняющие отображение веб-страниц. Такое разделение позволяет вынести вычислительно емкие операции с языковой моделью в главный процесс или отдельный фоновый процесс, не блокируя интерфейс пользователя; обмен данными между процессами осуществляется через механизм межпроцессного взаимодействия (IPC).

Локальная языковая модель подключается через библиотеку llama.cpp – открытую реализацию инференса трансформерных моделей на языке C++ с привязками для Node.js [5]. Поддерживается широкий спектр открытых трансформерных моделей семейств LLaMA, Mistral, Qwen и других [6]. Используется формат хранения GGUF, поддерживающий различные уровни квантования весов (от 8-битного до 2-битного), что позволяет сократить требования к оперативной памяти в 2–8 раз по сравнению с моделями в полной точности [7]. Пользователь самостоятельно выбирает модель необходимого размера, исходя из доступных вычислительных ресурсов: для устройств с 8 ГБ оперативной памяти подходят модели в диапазоне от 3 до 7 млрд параметров с агрессивным квантованием (Q4_K_M, Q5_K_S), для устройств с 16 ГБ и более – модели до 13 млрд параметров с менее агрессивным квантованием.

Запросы к модели формируются непосредственно из контекста веб-страницы: текущий выделенный пользователем фрагмент, заголовок страницы, метаданные документа, окружающий контекст. Контекст агрегируется в инструкцию для модели по шаблонам, специфичным для каждой операции (резюмирование, перевод, объяснение кода). Все запросы и ответы обрабатываются в рамках того же процесса приложения, без обращения к внешним сервисам. Опционально, при выборе пользователя, поддерживается работа с внешним AI-сервисом по совместимому с OpenAI протоколу – это позволяет применять более крупные модели на серверах, доверенных пользователю или организации.

Управление сетевым доступом реализовано через программный интерфейс session.webRequest движка Chromium, позволяющий перехватывать и фильтровать HTTP- и HTTPS-запросы на уровне приложения до их выполнения. Для каждого режима работы определяется набор правил фильтрации: в режиме «Только интрасеть» разрешаются исключительно запросы к адресам, входящим в заданные подсети организации; в режиме Smart внутренние и внешние адреса распознаются автоматически по DNS- и IP-признакам, и к каждому типу применяются собственные политики; в режиме «Интернет» правила не применяются. Подобный подход обеспечивает функциональность, недоступную браузерным расширениям: фильтрация выполняется не на уровне отдельной вкладки, а на уровне всего приложения, что исключает обход через системные API.

Для мобильной платформы используется среда Android с движком отображения WebView, обеспечивающая аналогичную многоплатформенную поставку приложения. Локальная языковая модель в мобильной версии подключается через нативные сборки

llama.cpp, скомпилированные для архитектуры ARMv8 с использованием Android NDK и инструкций NEON для ускорения векторных операций. С учетом ограниченного объема оперативной памяти мобильных устройств в данной редакции предусмотрена работа с моделями небольшого размера – от 1 до 3 млрд параметров с квантованием Q4_K_M, что обеспечивает соответствие занимаемой памяти типовым ограничениям мобильных конфигураций. Вопросы масштабирования решения охватывают как индивидуальное использование на одном устройстве, так и развертывание в рамках организации: предусмотрена централизованная конфигурация правил сетевого доступа и списка одобренных моделей через подписанный конфигурационный файл, что позволяет администратору единообразно настраивать рабочие места независимо от их числа и физического расположения.

РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ

В результате выполнения работы разработана браузерная оболочка Intra Browser, объединяющая функциональность стандартного веб-браузера на базе движка Chromium и модуля локального ИИ-анализа. Общий объем программного кода составил около 15 000 строк на языках JavaScript и TypeScript, реализована поддержка операционных систем Windows, Linux и Android. Установка приложения выполняется одним установочным пакетом без необходимости отдельной настройки среды исполнения; локальная языковая модель загружается отдельно по выбору пользователя из числа поддерживаемых GGUF-моделей.

Архитектурно приложение разделено на три уровня. Структурная схема приложения представлена на рисунке 1.

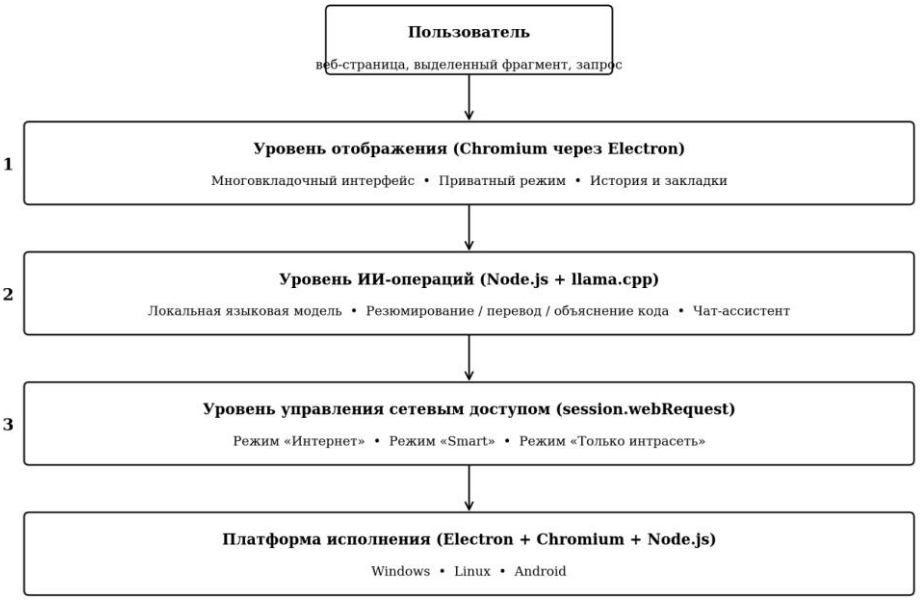

Рисунок 1 – Архитектура браузера Intra Browser

Первый уровень – уровень отображения, реализованный на базе движка Chromium через Electron. На этом уровне обеспечивается многовкладочный интерфейс пользователя, режим приватного просмотра (вкладки без сохранения истории и cookies), базовые функции навигации, управление историей и закладками. Интерфейс пользователя реализован средствами HTML, CSS и JavaScript; для обеспечения единообразия отображения на разных платформах используются общие стилевые правила.

Второй уровень – уровень ИИ-операций, обеспечивающий взаимодействие с локальной языковой моделью. При выделении пользователем фрагмента текста на странице доступны команды «Резюмировать», «Перевести», «Объяснить код», вызываемые из контекстного меню. Параллельно функционирует панель чат-ассистента, в которую пользова-

тель может направить произвольный запрос; в качестве контекста к запросу автоматически передается информация о текущей странице (заголовок, URL, выделенный фрагмент). Запрос формируется в главном процессе и передается в инстанс llama.cpp, исполняемый в отдельном дочернем процессе для обеспечения отзывчивости интерфейса; ответ возвращается в процесс рендеринга через IPC и отображается потоково по мере генерации токенов.

Третий уровень – уровень управления сетевым доступом, реализующий три режима работы: «Интернет» (полный доступ без ограничений, подходит для домашнего использования), Smart (автоматическое распознавание внутренних адресов организации с применением соответствующих правил – например, разрешение работы с внутренними ресурсами без проверки сертификатов), и «Только интрасеть» (полная блокировка внешних ресурсов, подходит для развертывания в учебных аудиториях и на рабочих местах с конфиденциальной информацией). Списки разрешенных и запрещенных адресов задаются администратором организации в конфигурационном файле и могут быть подписаны цифровой подписью для предотвращения несанкционированной модификации.

Пользователь имеет возможность выбора режима работы ИИ-ассистента: при использовании локальной модели данные не покидают пределов устройства; альтернативно, при наличии подключения к сети, допускается интеграция с внешним AI-сервисом по совместимому с OpenAI протоколу через настраиваемый коннектор. Такой подход обеспечивает гибкость применения и одновременно сохраняет за пользователем полный контроль над передачей данных: каждое обращение к внешнему сервису требует явного подтверждения и сопровождается журнальной записью.

Отдельное внимание уделено вопросам оптимизации производительности локальной языковой модели на устройствах с ограниченными вычислительными ресурсами. Для определения оптимального уровня квантования весов проведены замеры на тестовом стенде офисного класса (Intel Core i5-1135G7, 8 ГБ оперативной памяти) с использованием модели Qwen2.5 размером 7 млрд параметров. При квантовании Q4_K_M зафиксированы следующие характеристики: размер файла модели около 4,4 ГБ, потребление оперативной памяти при загрузке порядка 5,2 ГБ, время отклика до появления первого токена 1,2–1,8 с, скорость генерации в среднем 9–12 токенов в секунду. Переход к более агрессивному квантованию Q3_K_M сокращает потребление оперативной памяти до 4,1 ГБ и повышает скорость до 13–16 токенов в секунду ценой заметного снижения качества ответов на сложных запросах. Менее агрессивное квантование Q5_K_M, напротив, повышает качество при увеличении объема модели до 5,1 ГБ и снижении скорости до 7–9 токенов в секунду. Для дополнительного повышения отзывчивости применены: загрузка весов модели через mmap, что позволяет избежать копирования в пользовательское пространство; кэширование KV-состояний между запросами в рамках одной сессии чата; ограничение длины контекстного окна по умолчанию до 4096 токенов с возможностью расширения до 8192 при наличии достаточного объема оперативной памяти; опциональная выгрузка отдельных слоев нейронной сети на графический процессор через API CUDA, Metal или Vulkan при их доступности.

Дополнительно разработан пользовательский интерфейс, объединяющий функции браузера, корпоративного поиска, локального ИИ-ассистента и планирования рабочего процесса. Общий вид стартовой страницы приложения представлен на рисунке 2.

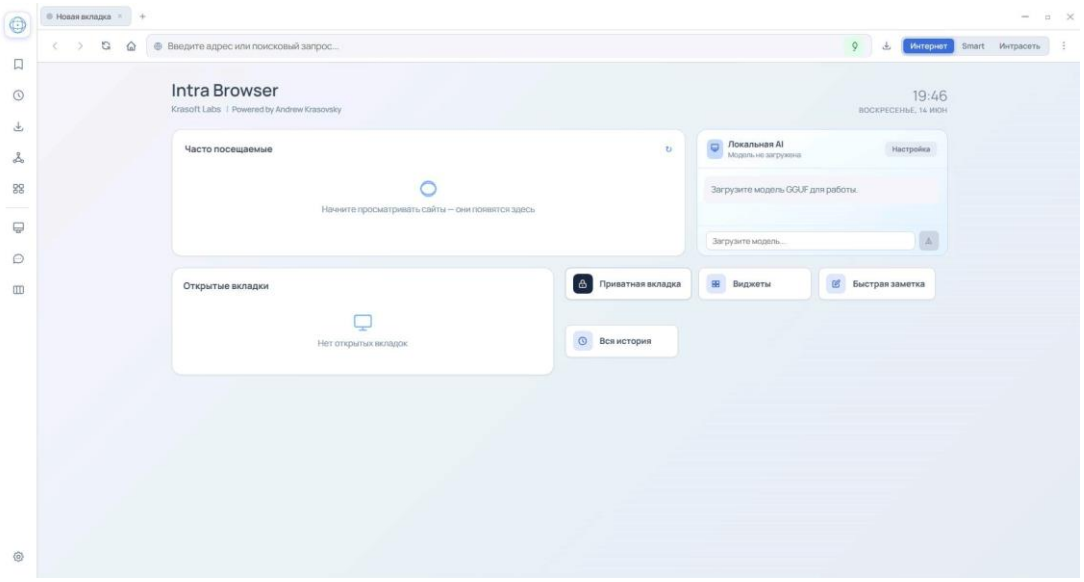

Рисунок 2 – Стартовая страница пользовательского интерфейса Intra Browser

Верхняя панель приложения содержит элементы управления вкладками, кнопки навигации (назад, вперед, обновить, переход на стартовую страницу), адресную строку с поддержкой поиска и ввода URL, индикатор состояния локальной модели, кнопку загрузок, переключатель режимов сетевого доступа («Интернет», Smart, «Интрасеть») и меню действий. Боковая панель слева предоставляет быстрый доступ к закладкам, истории посещений, журналу загрузок, контактам, виджетам, рабочему столу, чат-ассистенту, открытым колонкам и настройкам приложения.

Центральная область стартовой страницы организована как корпоративный рабочий стол. В ее состав входят: блок «Часто посещаемые», в котором отображаются наиболее посещаемые сайты и сервисы; блок «Открытые вкладки» с превью активных сессий; блок «Локальная AI», обеспечивающий загрузку и настройку GGUF-моделей через единое поле управления; кнопки быстрых действий («Приватная вкладка», «Виджеты», «Быстрая заметка», «Вся история»); информационный блок с текущим временем и датой. Подобная компоновка интерфейса ориентирована на сценарии корпоративной и учебной работы, в которых пользователь регулярно переключается между браузерными задачами и взаимодействием с локальным ИИ-ассистентом.

Отдельным результатом является спроектированная архитектура подсистемы децентрализованного обмена файлами (mesh-сеть) между устройствами в локальной сети организации. Подсистема предназначена для обмена файлами и текстовыми сообщениями без участия центрального сервера и применима в изолированных средах. В качестве кандидатов транспортного уровня рассмотрены технологии WebRTC (для прямой передачи между двумя узлами) и libp2p (для построения масштабируемой mesh-топологии). На момент представления работы реализация подсистемы находится в стадии проектирования: завершены анализ требований и выбор архитектурных решений, реализация планируется в следующей версии.

Для оценки результатов проведено сравнение с существующими решениями. В качестве объектов сравнения выбраны Google Chrome, Mozilla Firefox и Brave Browser как наиболее распространенные браузеры общего назначения. Сводные результаты сравнения по ключевым критериям представлены в таблице.

Из приведенных данных следует, что разработанное решение совмещает свойства, не реализованные одновременно ни в одном из общедоступных браузеров: локальное выполнение ИИ-операций в едином процессе с движком отображения, контекстная интеграция с содержимым страницы и встроенный механизм управления сетевым доступом на уровне приложения. Это подтверждает практическую и научную значимость предложенного подхода.

Таблица – Сравнение Intra Browser с существующими браузерами

Критерий	Chrome	Firefox	Brave	Intra Browser
Локальный ИИ без подключения к сети	нет	нет	нет	да
ИИ-операции над выделенным контентом страницы	частично (расширения)	частично (расширения)	да (облачно)	да (локально)
Управление сетевым доступом на уровне приложения	нет	нет	нет	да (3 режима)
Поддержка корпоративной интрасети без внешних сервисов	нет	нет	нет	да
Кроссплатформенность (Windows / Linux / Android)	да	да	да	да

Прикладные сценарии использования включают: применение в учебных аудиториях с ограничением доступа к ресурсам, не относящимся к учебному процессу (например, при проведении контрольных работ); работу сотрудников организаций с конфиденциальными документами без риска передачи данных в сторонние сервисы; поддержку рабочих процессов в условиях ограниченного или отсутствующего доступа к внешней сети – на изолированных рабочих станциях или в полевых условиях.

ЗАКЛЮЧЕНИЕ

В работе представлен прототип кроссплатформенной браузерной оболочки Intra Browser на базе движка Chromium с интегрированным модулем локального ИИ-анализа веб-страниц. Реализованная архитектура на базе Electron, llama.cpp и нативных Android-компонентов позволяет выполнять интеллектуальные операции над содержимым страницы без передачи данных в сторонние сервисы и с возможностью гибкого управления сетевым доступом на уровне приложения, что особенно актуально для образовательных и корпоративных сред с повышенными требованиями к конфиденциальности информации.

Научная значимость работы состоит в обосновании возможности рассматривать браузерную оболочку как платформу для локальных интеллектуальных операций, а не исключительно как клиент удаленных сервисов, и в архитектурных решениях, обеспечивающих такую интеграцию: разделении на главный и рендер-процессы, использовании межпроцессного взаимодействия для асинхронного обращения к языковой модели, перехвате сетевого трафика на уровне session.webRequest. Практическая значимость подтверждается работающим прототипом, поддерживающим операционные системы Windows, Linux и Android, экспериментально установленными характеристиками производительности при различных уровнях квантования модели и встроенной системой управления сетевым доступом.

Дальнейшие направления исследования включают завершение реализации подсистемы децентрализованного обмена данными в локальной сети на базе WebRTC и libp2p, дальнейшую оптимизацию производительности локальной языковой модели на устройствах с ограниченными вычислительными ресурсами (в том числе исследование применимости моделей с агрессивным квантованием уровня Q3 и Q2 и автоматический подбор уровня квантования по доступным ресурсам), а также добавление режима чтения и темной темы оформления интерфейса.

СПИСОК ЛИТЕРАТУРЫ

1. Vaswani, A. Attention is all you need / A. Vaswani, N. Shazeer, N. Parmar [et al.] // Advances in Neural Information Processing Systems. – 2017. – Vol. 30. – P. 5998–6008.

2. Brown, T. B. Language models are few-shot learners / T. B. Brown, B. Mann, N. Ryder [et al.] // Advances in Neural Information Processing Systems. – 2020. – Vol. 33. – P. 1877–1901.

3. Bommasani, R. On the opportunities and risks of foundation models / R. Bommasani, D. A. Hudson, E. Adeli [et al.] // arXiv preprint arXiv:2108.07258. – 2021.

4. Electron Documentation [Электронный ресурс]: офиц. сайт фреймворка Electron. – URL: https://www.electronjs.org/docs (дата обращения: 01.05.2026). – Текст: электронный.

5. llama.cpp [Электронный ресурс]: репозиторий проекта на GitHub. – URL: https://github.com/ggml-org/llama.cpp (дата обращения: 01.05.2026). – Текст: электронный.

6. Touvron, H. LLaMA: Open and efficient foundation language models / H. Touvron, T. Lavril, G. Izacard [et al.] // arXiv preprint arXiv:2302.13971. – 2023.

7. Kurt, U. Which Quantization Should I Use? A Unified Evaluation of llama.cpp Quantization on Llama-3.1-8B-Instruct / U. Kurt // arXiv preprint arXiv:2601.14277. – 2026.

DEVELOPMENT OF A CROSS-PLATFORM USER WEB BROWSER WITH INTEGRATED AI-BASED PAGE ANALYSIS

A. K. Krasovskii, 3rd year student
E-mail: andrew.krasovsky39@gmail.com
Kaliningrad State Technical University

A. N. Ilyashov, senior lecturer
E-mail: andrej.ilyashov@klgtu.ru
Kaliningrad State Technical University

This article presents the development of a cross-platform browser shell called Intra Browser, based on the Chromium rendering engine, with an integrated module of local artificial intelligence for the analysis of web page content. The proposed solution is based on the Electron framework and runs a language model directly on the user's device through the llama.cpp library, providing summarization, translation, and code explanation features for selected page content without sending data to remote services. The application also implements a built-in network access management subsystem with three operating modes – Internet, Smart, and Intranet-only – designed for use in organizations with strict information security requirements. The browser supports Windows, Linux, and Android operating systems. The paper describes the test bench configuration, the inter-process communication mechanisms, the handling of web page context, the principles of working with local language models in GGUF format, performance measurements at different quantization levels, and the implementation of network traffic filtering through the session.webRequest interface of the Chromium engine. A comparison with the most widely used existing browsers (Chrome, Firefox, Brave) has been provided. The total volume of the implemented prototype is approximately fifteen thousand lines of source code in JavaScript and TypeScript.

Keywords: *web browser, browser shell, local artificial intelligence, language model, Electron, Chromium, llama.cpp, GGUF, cross-platform development, data privacy, corporate environment.*