ChatGPT

Децентрализованные вычислительные системы без серверного бэкенда: архитектура и принципы

Введение

Децентрализованные вычисления – это подход, при котором приложения обходятся без традиционного серверного бэкенда и централизованных дата-центров. Вместо единого сервера используется совокупная мощность множества узлов (устройств) в сети, зачастую прямо на стороне клиентов [1]. Такой подход устраняет единую точку отказа и контроля, распределяя вычисления по узлам сети. Современные веб-технологии позволяют создавать сложные приложения, выполняющиеся полностью на клиентской стороне – весь код работает в браузере пользователя, а не на удалённом сервере [2]. Это даёт ряд преимуществ: отсутствие затрат на поддержание сервера, упрощённое развёртывание (достаточно выложить статические файлы), а также повышенную устойчивость и приватность, так как данные не покидают устройство пользователя [3] [4]. Децентрализация вычислений значительно повышает **масштабируемость** (за счёт суммирования ресурсов большого числа узлов) и **устойчивость** системы – отсутствие единого центра делает сеть более живучей к сбоям и атакам [5]. Ниже рассматриваются технические принципы такой архитектуры, с упором на клиентскую сторону и современные инструменты вроде **TypeScript**, **WebGPU**, **WebAssembly**, P2P-протоколов и распределённых сетей GPU.

Клиент-ориентированная архитектура без сервера

В архитектуре “только фронтенд” всё приложение выполняется на устройстве пользователя. Браузеры сегодня достаточно мощны, чтобы обрабатывать бизнес-логику и хранить данные локально. Для хранения данных используются, например, **LocalStorage** или **IndexedDB** – встроенные хранилища браузера, позволяющие сохранять настройки, кеш или даже целые базы данных на стороне клиента [6] [7]. Благодаря этому пользователь может работать с приложением даже офлайн, а данные никогда не отправляются на сторонний сервер без необходимости, обеспечивая **конфиденциальность** (например, чувствительные фотографии или документы остаются локально) [8].

Для взаимодействия и синхронизации информации между клиентами применяются **Peer-to-Peer (P2P)** связи. В веб-среде основой P2P-коммуникаций служит **WebRTC** – технология, позволяющая браузерам устанавливать прямое соединение друг с другом для обмена данными минуя сервер [9]. Через WebRTC (в частности, его DataChannel API) браузеры могут передавать произвольные данные или медиа напрямую. Важно отметить, что для инициации P2P-соединения обычно нужен этап сигнализации – обмен первоначальными метаданными через промежуточный сигнальный узел. Однако такой сигнальный сервер крайне лёгкий и не участвует в передаче основного трафика – он лишь помогает двум браузерам "найти" друг друга [10] [11]. После установления соединения обмен идёт непосредственно между клиентами (шифрованно), что устраняет задержки и расходы на проксирование трафика через бэкенд [12].

Анонимность и безопасность. В P2P-схеме данные не проходят через центральный сервер, поэтому разработчик не получает доступ к передаваемому контенту, будь то видео-звонок или текст переписки [13]. Пользователям не требуется регистрация – приложение может функционировать без учётных записей, не храня cookies и не отслеживая действия, что привлекает аудиторию, ценящую приватность [14]. При этом следует учитывать, что P2P-режим раскрывает IP-адреса узлов собеседников, поэтому дополняют анонимность часто за счёт VPN или маршрутизации через анонимные сети при необходимости [15]. В целом, фронтенд-ориентированная архитектура сокращает **точки отказа** (приложение продолжит работать, пока доступен хотя бы пир или сеть доставки статических файлов) и избавляет разработчика от многих задач сопровождения инфраструктуры.

WebAssembly и WebGPU на стороне клиента

Ключевыми технологиями, расширяющими возможности клиентских вычислений, являются **WebAssembly (WASM)** и **WebGPU**. WebAssembly – это низкоуровневый бинарный формат, исполняемый в браузере с почти нативной скоростью. Он позволяет запускать на клиенте код, написанный на разных языках (C/C++/Rust и др.), с производительностью близкой к приложению вне браузера, оставаясь в песочнице безопасности. Скомпилированные WASM-модули могут выполнять тяжёлые алгоритмы значительно быстрее, чем чистый JavaScript. Уже сегодня существуют, например, WASM-библиотеки для машинного обучения, обработки изображений, шифрования и прочих ресурсоёмких задач – их можно вызывать из TypeScript-кода приложения. Применение таких библиотек в браузере даёт эффект, когда **даже сложная обработка (например, фильтрация или распознавание изображения нейросетью) выполняется локально, без отправки на сервер** [8]. Это не только защищает данные пользователя (они никуда не передаются), но и экономит средства – разработчику не нужно оплачивать мощные серверы или внешние API для ML, если вычисления производятся на устройстве клиента [16].

WebGPU – относительно новый веб-стандарт, предоставляющий браузерным приложениям высокоуровневый доступ к графическому процессору (GPU) для параллельных вычислений. В отличие от устаревшего WebGL (ориентированного на 3D-графику), API WebGPU подходит для общего параллельного программирования на GPU – от рендеринга до линейной алгебры и нейросетей. С помощью WebGPU веб-приложение на TypeScript может выполнять массовые вычисления на видеокарте пользователя, получая значительное ускорение. Например, операции над матрицами, рендеринг сложной графики или расчёт нейронных сетей выполняются на порядки быстрее благодаря тысячам параллельных потоков GPU. Это открывает дорогу **клиентским AI-возможностям** – вплоть до запуска компактных моделей глубокого обучения прямо в браузере. Появились экспериментальные реализации, где большие модели (например, языковые модели или обработка изображений) запускаются с задействованием WebGPU, демонстрируя, что современное пользовательское устройство может взять на себя вычислительную нагрузку, ранее требовавшую облачных GPU. Конечно, необходимо убедиться, что у конечного пользователя достаточно мощное устройство, иначе выполнение моделей будет медленным [17]. Тем не менее, тренд таков, что **производительность клиентских устройств выросла** – они способны выполнять задачи, которые раньше приходилось делегировать серверу (генерация отчетов, сортировка больших наборов данных, рендеринг страниц), снижая задержки за счёт отсутствия сетевых запросов [18].

Комбинация WebAssembly и WebGPU с языками вроде TypeScript позволяет реализовать на фронтенде **логические модули любой сложности** – от криптографических операций и симуляций до машинного обучения – и выполнять их внутри браузера, изолированно и эффективно. Таким образом, браузер превращается в полноценную вычислительную платформу, а сеть выступает лишь транспортом для обмена уже обработанными данными или моделями.

Выполнение AI-моделей на стороне клиента

Отдельно стоит рассмотреть запуск искусственного интеллекта (AI) и моделей машинного обучения на клиенте. Ранее для таких задач практически всегда требовался сервер (например, для работы с TensorFlow/PyTorch моделями на GPU). Сейчас же появились инструменты, позволяющие обучать и применять модели **на стороне пользователя**.

Пример – библиотека **TensorFlow.js**, которая позволяет загружать предобученные нейросети и выполнять их прямо в браузере, используя либо ускорение через WebGL, либо через WebGPU (в новых версиях). Аналогично, **ONNX Runtime Web** и другие WebAssembly-ориентированные фреймворки позволяют выполнять предобученные модели (формата ONNX, тензорные вычисления) внутри браузера с ускорением. Уже демонстрировались приложения, где распознавание изображений, обработка естественного языка и даже генеративные модели работают локально. Так, **распознавание текста (OCR)** на фотографии можно реализовать с помощью нейросети, загруженной в браузер, – фотография не отправляется никуда во внешнюю среду, а распознаётся полностью на устройстве пользователя. Это критично для конфиденциальности, например, в приложениях обработки медицинских снимков: данные пациента остаются только у него [8]. Ещё пример – стилизация изображения или фильтры, работающие через нейросеть: пользователь выбирает фото, и в браузере применяется модель, например, преобразующая стиль (как Prisma, но без облака).

С появлением WebGPU стало возможным запускать даже тяжёлые модели, требующие параллельных вычислений. К примеру, модель **Stable Diffusion** (генерация изображений по текстовому описанию) была портирована для браузера с использованием WebGPU – энтузиасты показали, что современный ноутбук способен сгенерировать изображение за считанные секунды, выполняя инференс нейросети локально. Также есть прецеденты запуска языковых моделей (пусть и облегчённых версий) через WebAssembly: например, небольшие модели типа GPT-2 или LLaMA 7B в сжатом виде могут работать на стороне клиента, обеспечивая автономную работу простого чат-бота без обращения к API.

Преимущества **AI on the Edge** (на клиенте) – это, помимо приватности данных и отсутствия задержек сети, ещё и **экономия** на инфраструктуре для разработчика. Нет нужды платить за использование облачных AI-сервисов или содержать свой сервер с GPU: достаточно однажды загрузить модель в фронтенд. Конечно, остаются вызовы: объём моделей ограничен памятью устройства, а длительные вычисления могут нагружать процессор/GPU пользователя (влияя, например, на заряд батареи). Поэтому в рамках децентрализованной архитектуры часто комбинируют подходы: менее требовательные задачи выполняют на клиенте, а очень тяжёлые – делегируют во внешнюю распределённую сеть вычислений. Например, приложение может пытаться выполнить ML-запрос локально, а если устройству не хватает мощности, обратиться к децентрализованному облаку GPU (о чём далее).

P2P-протоколы для распределённых вычислений

Чтобы множество клиентов могло совместно выполнять вычислительные задачи, нужны эффективные P2P-протоколы и сетевые механизмы координации. **Без централизованного API** узлам необходимо обнаруживать друг друга, передавать задания и обмениваться результатами напрямую или через распределённые таблицы маршрутизации.

Обнаружение узлов. В децентрализованных сетях часто применяется распределённая хеш-таблица (**DHT**) – это механизм, позволяющий узлам находить нужные ресурсы или других

участников по ключу (например, контент-адресу). Проект **IPFS** (межпланетная файловая система) использует DHT для нахождения узлов, хранящих контент по его хешу, без центрального сервера. Аналогично, вычислительная задача может быть описана неким идентификатором, и узлы-исполнители могут быть найдены через DHT по требуемому ресурсу (скажем, нужен узел с GPU определённого класса или узел, владеющий определёнными данными).

Обмен данными. Помимо WebRTC в браузерах, существуют и другие P2P-технологии обмена: например, **WebTorrent** – реализация протокола BitTorrent, работающая в браузере через WebRTC. Она позволяет раздавать и скачивать файлы по P2P прямо на фронтенде [19]. В контексте вычислений, крупные наборы данных или модели могут распространяться через подобные механизмы: узлы поделятся частями данных напрямую, разгружая какой-либо один сервер (аналогично тому, как торренты ускоряют раздачу за счёт параллельного получения частей от разных пиров) [20]. Для синхронизации состояний можно применять **pub/sub**-схемы поверх P2P: например, протокол **libp2p** предоставляет модуль gossipsub, через который узлы могут рассылать сообщения группе подписчиков – это удобно для широковещательных обновлений или объявлений новых задач.

WebRTC DataChannels – важный инструмент для браузерных вычислительных сетей. Через них можно организовать обмен сообщениями между браузерами не только один-на-один, но и в топологиях типа "сеть друзей". Однако масштабируемость WebRTC P2P-схемы ограничена: каждый браузер не сможет поддерживать очень много одноранговых соединений одновременно (из-за нагрузки на CPU и сети) [21] [22]. Поэтому в реальных системах для большого числа участников применяют либо иерархические сети (кластеры, суперузлы), либо частично децентрализованные решения (например, федеративные сервера). Тем не менее, для средних масштабов (десятки узлов) прямое P2P-взаимодействие работает успешно – существуют прототипы видеоконференций и чат-приложений, где браузеры обмениваются медиаданными напрямую без промежуточного сервера [23] [24].

Прозрачность и идентификация. В распределенной сети каждый узел может иметь криптографический идентификатор (пара ключей), чтобы взаимно аутентифицировать сообщения без централизованного удостоверяющего центра. Стандарты децентрализованных идентификаторов (DID) и самоподписанных сертификатов позволяют узлам доверять транзакциям, если они подписаны знакомым ключом. Это особенно полезно, когда узлы договариваются о выполнении кода: задача и результат могут быть подписаны исполнителем и проверяющим. Также, в P2P сетях часто ведётся **распределённый лог** событий (например, на основе блокчейна или просто общий журнал), чтобы участники могли убедиться, что все видят одну и ту же картину (какие задачи поставлены, кем выполнены и т.д.).

В итоге P2P-протоколы обеспечивают коммуникационный фундамент: **распространение данных, заданий и результатов между клиентами без центрального посредника**. Далее рассмотрим, как на базе этой коммуникации строятся целые сети обмена вычислительными ресурсами.

Распределённые GPU-сети и вычислительные рынки

Для выполнения требовательных задач (рендеринг, ML-тренинг, научные расчёты) в децентрализованной среде появились специализированные **сети распределённых вычислений**, часто называемые децентрализованным облаком. Они объединяют множество узлов с мощным оборудованием (GPU, ASIC и пр.) в единую платформу, где вычислительные мощности могут арендоваться или предоставляться в аренду P2P-образом.

Filecoin и Compute-over-Data. Проект Filecoin известен как децентрализованное хранилище данных, но на его базе развивается и направление распределённых вычислений. Идея в том, что огромные объёмы данных (например, датасеты для ИИ) уже хранятся на узлах Filecoin, и выгоднее **привезти вычисления к данным**, чем пересылать данные к вычисляющему узлу [25] [26]. Для этого существует инициатива *Compute over Data* и платформа **Bacalhau**. Bacalhau позволяет запускать задачи (Docker-контейнеры или WebAssembly-задачи) непосредственно на узлах, где лежат необходимые данные, выполняя параллельные вычисления рядом с местом хранения [27] [28]. Это снижает задержки и стоимость, так как не тратится трафик на перемещение больших файлов. В сети Bacalhau узлы-вычислители (Compute Providers) регистрируются и принимают задания, а результаты могут быть проверены и сохранены обратно в распределённое хранилище. Примечательно, что Filecoin с запуском виртуальной машины FVM получил возможность брокеровать вычислительные ресурсы, **стимулировать выполнение вычислений и даже криптографически доказывать корректность результатов** на блокчейне [29]. То есть смарт-контракты Filecoin могут выступать децентрализованным координатором, распределяя задачи по storage-провайдерам, отслеживая их выполнение и награждая за успешные вычисления.

Render Network. Это одна из первых реализованных распределённых GPU-сетей, сфокусированных на рендеринге графики. Созданная компанией OTOY, **Render Network** соединяет владельцев видеокарт (GPU-провайдеров) с художниками и аниматорами, которым нужно отрендерить сцены или видео. По сути, это **пиринговый рынок GPU-вычислений**: владельцы сдаёт в аренду простаивающие мощности, а заказчики оплачивают рендеринг с помощью токена RNDR [30] [31]. Сеть стартовала в 2020 году и к 2023 выделилась в отдельный фонд Render Network Foundation [32]. Архитектурно, Render Network использует блокчейн для учёта работ и расчетов между сторонами. Узлы (рендер-ноды) получают задания, просчитывают кадры или 3D-сцены с помощью ПО OTOY (например, OctaneRender), и отправляют результаты обратно. Система обеспечивает проверку корректности (например, путём небольших контрольных тасков или репутации узлов) и тем самым гарантирует, что клиент получит верный отрендеренный продукт, прежде чем перевести оплату. Сеть Render стала **пионером децентрализованного GPU-рендеринга**, фактически агрегируя глобальную избыточную GPU-мощность для нужд индустрии CGI. Примечательно, что теперь эту же сеть планируют использовать для задач AI: так, партнёрство Stability AI и Render Network направлено на масштабирование обучения и инференса моделей ИИ на массово распределённых GPU по доступной цене [33].

Golem Network. Golem – один из ранних проектов (анонсирован еще в 2016), стремившихся создать «децентрализованный суперкомпьютер». Он позволяет выполнять произвольные вычислительные задачи в сети участников: от рендеринга CGI (как был демонстрационный кейс проекта) до научных расчетов. Архитектура Golem подразумевает, что заказчик разбивает задачу на части (задачи могут быть в контейнерах или скомпилированных бинарях), выбирает исполнителей из сети и рассылает им работу. Результаты затем собираются и агрегируются. Для обеспечения честности Golem использует комбинацию методик: **репликация вычислений** (несколько узлов могут выполнить одну задачу, результаты сверяются) и **стимулы/штрафы** через внутренний токен GNT (теперь GLM). Если узел мошенничает (выдаёт неверный результат), он рискует не получить плату или потерять залог, а его репутация падает. Golem изначально работал поверх Ethereum (для контрактов оплаты), но со временем были оптимизированы внецепочечные компоненты для поиска узлов и передачи данных. Сегодня Golem – действующая сеть с сотнями узлов, подходящая для задач, допускающих параллелизм. Это хороший пример **общего децентрализованного пула CPU/GPU**, не привязанного к одному виду задач.

Распределённые сети для AI. В последние годы появилось несколько специализированных проектов, нацеленных на распределённое обучение и инференс ML-моделей. Один из ярких – **Gensyn**, позиционируемый как “network for machine intelligence”. Gensyn строит открытый протокол, способный подключить *любые устройства в мире в единую сеть для обучения моделей* без посредников и ограничений [34] [35]. Они решают несколько ключевых проблем, стоящих перед такой сетью: совместимость вычислений на разнородных устройствах, **доверенная проверка результатов**, эффективное расшаривание нагрузки и децентрализованная координация участников [36]. В частности, для проверки Gensyn разработал систему **Replicate Ops** (RepOps) для битового воспроизведения результатов на разных устройствах – это позволяет арбитражу сети убедиться, что разные исполнители получили один в один совпадающие результаты на одной задаче [37] [38]. Кроме того, Gensyn исследует методы **trustless verification** (доверенного контроля) с минимальными накладными расходами, например, разбивая ML-тренинг на этапы, которые можно проверять, и используя криптографические доказательства. Координация в Gensyn осуществляется через кастомный блокчейн-роллап (специализированный L2), где регистрируются задания, результаты и вознаграждения [39]. Фактически, это гибрид: интенсивные вычисления происходят оффчейн на узлах, но их результаты подтверждаются и заверяются на блокчейне. Помимо Gensyn, стоит упомянуть проект **Render Network for AI** (коллаборация Render с партнёрами) и **Hivemind** (масштабируемый P2P обучения языковых моделей от разработчиков EleutherAI), а также проекты типа **Petals** – сеть для совместного хранения и инференса больших языковых моделей, где каждый узел хранит часть слоёв модели и обслуживает запросы локально. Эти инициативы указывают на эволюцию: обучение ИИ может выйти за пределы дата-центров и распределиться по большим сетям, если удастся обеспечить достаточно быструю связь и согласованность.

Другие примеры. В экосистеме Web3 также существуют: **Fluence** – децентрализованная серверлесс-платформа, на которой разработчики могут размещать peer-to-peer приложения и функции без единого сервера [40]. Провайдерами Fluence выступают и профессиональные дата-центры, и частные компьютеры – любой, кто хочет предоставить вычислительные мощности. Fluence предлагает свой язык Aqua для описания распределённой логики и обеспечивает *безкоординаторное выполнение* – без единого узла, управляющего запросом [41]. Также примечателен проект **iExec RLC**, предлагающий децентрализованный облачный маркетплейс, где вычислительные задачи могут выполняться либо в доверенной среде (Trusted Execution Environment на узлах) для гарантии результатов, либо через проверяемое многократное исполнение. **Akash Network** – децентрализованный рынок, где можно арендовать вычислительные ресурсы (аналог AWS EC2) за крипто-токены, разворачивая контейнеры на независимых провайдерах. Все эти сети демонстрируют, что задачи, ранее решаемые только центральным облаком, теперь могут распределяться между множеством участников, конкурирующих за выполнение – что снижает цены и устраняет монополию крупных игроков [42] [43].

Координация распределённых вычислений

Одна из самых сложных задач в отсутствии централизованного сервера – **координация**: как направить нужное вычисление на подходящий узел, собрать ответы и выдать результат пользователю, сохранив при этом согласованность. В централизованных системах эту роль выполняет сервер (планировщик, оркестратор). В децентрализованных – приходится полагаться на протоколы и алгоритмы.

Маркетплейсы и блокчейны. Много распределённых вычислительных сетей используют блокчейн для координации заявок и оплаты. Например, заказчик публикует на смарт-контракте *задачу* с указанием вознаграждения. Исполнители (воркеры) просматривают глобальную очередь

задач и берут в работу те, которые подходят их ресурсам. Контракт гарантирует резервирование оплаты и может выступать арбитром спора. В Golem именно так – задание и хеш предполагаемого результата регистрируются в сети, исполнитель вычисляет и возвращает результат, после чего получает оплату. Если результат оспорен – инициируется арбитраж (например, повторный расчёт на проверочном узле). Новые подходы, как у Gensyn, выносят эту механику на отдельный sidechain/rollup, оптимизированный по пропускной способности под частые вычислительные сделки [39].

Репликация и консенсус. Без доверия к отдельным узлам приходится либо **дублировать выполнение** на нескольких узлах, либо внедрять криптографические методы проверки (о них далее). В случае дублирования сеть должна собрать результаты, сравнить их и решить, какой считать истиной. Например, Fluence реализует стратегии *кворума*: можно задать, что результат считается успешным, если его подтвердили, скажем, 2 из 3 узлов (мульти-исполнение) [44]. Это похоже на консенсус в BFT-системах: если большинство не скомпрометировано, они выдадут верный ответ, и меньшинство (возможный злоумышленник) будет проигнорировано. Такой подход увеличивает накладные расходы (нужно запустить задачу несколько раз), но повышает надёжность исполнения в недоверенной среде [45] [46].

Choreography vs Orchestration. Интересной концепцией Fluence являются *Cloudless Functions* – по сути, фрагменты кода, описывающие как задача должна путешествовать по сети узлов, какие данные куда передавать и как агрегировать ответы [47] [41]. Вместо центрального оркестратора, разработчик описывает *хореографию*: например, "отправь запрос на 5 случайных узлов, дождись трёх ответов, выбери самый частый и передай следующему этапу". Исполнение такой функции происходит *координатор-менее* (без единого управляющего) – каждый узел следует сценарию, заложенному в запросе. Язык Aqua позволяет задать такие распределённые сценарии, и они выполняются автономно, включая повтор запросов при таймаутах, слияние результатов и пр. [48] [44]. Конвергенция достигается гарантиями протокола: Aqua обеспечивает отсутствие replay-атак и MITM – нельзя захватить или подделать ответ пира без обнаружения [49]. Таким образом, координация частично перекладывается на самих участников – они **соблюдают протокол**, который предписывает логику взаимодействия.

Поиск исполнителей. Важно выбрать, какие узлы будут выполнять задачу. В децентрализованных сетях могут использоваться каталоги возможностей: например, узлы объявляют свои ресурсы (количество CPU, объём памяти, наличие GPU). Это объявление хранится либо на блокчейне (в реестре), либо распространяется по DHT. Когда приходит задача, требующая, скажем, GPU с 8 ГБ памяти, заказчик или протокол находит соответствующих кандидатов через фильтрацию реестра. Filecoin, к примеру, упоминалось, что у storage-провайдеров уже есть вычислительное оборудование рядом с данными – и compute-задачи могут быть распределены среди них [50]. Системы вроде **Web3Mine** и **Io.net** специализируются на координации физических (офлайн) ресурсов: они помогают коллективно управлять распределёнными датацентрами, следить за их показателями и привлекать их к задачам [51]. В случае Io.net, это осуществляется через свою платформу и Solana-блокчейн: узлы регистрируются, получают рейтинг надёжности и могут быть выбраны смарт-контрактом для развертывания кластера ML по запросу клиента [52] [53].

Оптимизация коммуникации. Для эффективной координации крупномасштабных вычислений на ненадёжных соединениях разрабатываются специальные алгоритмы. Gensyn, например, предлагает метод **NoLoCo** – gossip-алгоритм, заменяющий глобальную синхронизацию параметров при распределённом обучении на постепенное разнесение обновлений по сети, что лучше подходит для гетерогенных и медленных сетей [54]. Другой пример – **SkipPipe**: метод, уменьшающий простои при конвейерном параллелизме обучения и допускающий сбои до 50%

узлов без остановки процесса [55]. Эти разработки направлены на то, чтобы в условиях распределённости и возможных потерь соединения всё же добиться приемлемой эффективности и скорости.

В сумме, координация в децентрализованных вычислительных системах достигается сочетанием *глобального консенсуса* (когда нужно зафиксировать факт выполнения и выплат, обычно через блокчейн) и *локальной договорённости* (протоколы между участвующими узлами). Эта двухуровневая схема позволяет избавиться от постоянного центрального управляющего сервера, сохранив при этом управляемость и предсказуемость системы.

Механизмы доверия и проверка результатов

Когда вычисления выполняются на сторонних узлах, встаёт вопрос доверия: как убедиться в корректности результата, не полагаясь “на честное слово” удалённого исполнителя? В децентрализованных системах для этого используются **криптографические и экономические механизмы.**

Крипто-экономические стимулы. Многие протоколы вводят для узлов необходимость ставить залог (stake) или иметь репутацию. Если узел предоставляет неправильный результат, он рискует потерять заложенные средства и подорвать свою репутацию, что в будущем лишит его возможностей заработать. Таким образом, добросовестное поведение поощряется, а мошенничество делает экономически невыгодным. Например, в Filecoin сами хранилища заложили collateral и доказывают своё благочестие регулярными доказательствами хранения – подобный принцип может распространяться и на вычисления: узел мог бы периодически предоставлять доказательства правильного выполнения определённых бенчмарков или небольших проверочных задач, иначе его stake сгорает.

Репликация и голосование. Самый прямолинейный способ проверки – **запросить вычисление у нескольких независимых узлов**. Если большинство (кворум) возвращает одинаковый ответ, велика вероятность, что ответ верен, а отклонившиеся узлы (если такие есть) – ошибались или вредоносны [45]. В схемах византийского консенсуса (BFT) обычно предполагается, что доля честных узлов > 2/3, тогда консенсус достигается. Однако, дублирование работы удорожает процесс: считать одну и ту же задачу n раз – накладно, поэтому подобные подходы применяют избирательно, для критичных либо небольших вычислений (либо комбинируют: например, каждую сотую задачу проверяют вторым прогоном случайным образом). Тем не менее, репликация – эффективный способ для *детерминированных* задач. Впрочем, он не даёт 100% гарантии, а лишь сильно повышает стоимость атаки (злоумышленнику пришлось бы захватить многие узлы). Как отмечается в исследованиях, **избыточное вычисление** повышает стоимость для потребителя и не устраняет полностью риск, а только снижает его [46].

Детерминизм и воспроизводимость. Чтобы можно было проверить результат, вычисление должно быть детерминированным – т.е. на любых машинах при тех же входных данных давать тот же выход (вплоть до битового соответствия). Разные архитектуры и GPU могут давать небольшие различия (например, из-за порядка операций с плавающей запятой). Поэтому проекты типа Gensyn вкладываются в **RepOps – библиотеки для битового детерминизма**: например, использовать ограниченную точность, фиксировать сиды генераторов случайных чисел и пр., чтобы два узла гарантированно пришли к идентичному побитовому результату [38]. Тогда проверка сводится к пересчёту – если хеш результата совпал с присланным, значит узел не мухлевал. Gensyn разработал компонент **Verde** – арбитражную систему, которая использует такую

воспроизводимость: спорные моменты решаются перепроверкой работы (возможно, на нескольких узлах) до тех пор, пока все не согласятся с исходом [37] [38]. Такой метод напоминает проект TrueBit (для Ethereum), где задача выполняется вне блокчейна, но если кто-то из участников усомнился в результате, он инициирует *верификационную игру*: задача делится на подзадачи, которые перепроверяются, и жульничающий исполнитель в итоге изобличается и штрафуется.

Zero-knowledge proofs (ZK) для вычислений. Криптографическим "святым Граалем" верификации является подход, когда сам исполнитель генерирует доказательство корректности выполнения программы, которое можно быстро проверить. Zero-knowledge proof позволяет доказать, что *некоторая программа P с входом X выдала результат Y*, при этом проверяющий тратит на проверку *намного меньше ресурсов*, чем потребовало само исполнение P. Идея в том, что узел-исполнитель вместе с результатом посылает компактное доказательство (например, SNARK), и получателю не нужно заново выполнять всю задачу – достаточно проверить доказательство, что занимает считанные миллисекунды. Более того, **доказательство нулевого разглашения** может подтвердить корректность, не раскрывая никаких внутренних данных (полезно, если входы/алгоритм приватные) [56]. Применительно к ML это вылилось в направление *ZKML* – доказательства правильности вывода нейросети. Однако на текущий момент технологии ZK для общих вычислений очень ресурсозатратны. Генерация SNARK-доказательства сложной программы может в тысячи раз превышать по времени саму программу [57]. В частности, для даже средней нейросети с парой миллионов параметров сформировать доказательство инференса – огромное вычислительное усилие, и пока что подобное считается непрактичным (исключение – крайне простые модели или отдельные шаги). Сейчас исследования сосредоточены на оптимизации: ожидается, что эффективность генерации ZKP растёт ~на 3-4% в день от новых находок и улучшений алгоритмов [58]. Уже есть успешные демонстрации: например, доказательство выполнимости решателя Sudoku или простейшей нейросетевой классификации с помощью систем типа Groth16 или PLONK. Постепенно ZKP могут начать интегрироваться в проверку распределённых вычислений, особенно если удастся сократить стоимость. Тогда, как отмечают эксперты, это **позволило бы проверять корректность вычислений почти однозначно и без доверия**, даже для недетерминированных или очень сложных задач [59]. Но пока такие решения ограничены, в основном, фазой *инференса* ML (не обучения) и относительно небольшими моделями [60].

Полностью гомоморфное шифрование (FHE). Этот криптографический метод обеспечивает другую грань доверия – *приватность данных*. FHE позволяет производить вычисления над данными, остающимися зашифрованными. То есть клиент шифрует свой ввод, отправляет в облако, облако выполняет некоторую программу над шифрованными данными, получая зашифрованный результат, который клиент потом дешифрует. Облачный узел при этом **никогда не видит исходных данных в открытом виде**, а значит, не может их утечь или использовать не по назначению. Это крайне привлекательно для сценариев вроде медицинских вычислений на внешних мощностях или чувствительных бизнес-данных. Однако у FHE есть минус: выполняя операцию, удалённый узел может нарочно схалтурить и выдать случайный зашифрованный результат – клиент не сможет это обнаружить до расшифровки. То есть FHE сам по себе не решает задачу доверия к корректности, а только по сути делает ненужным доверие в плане конфиденциальности. Тем не менее, комбинация FHE и ZKP рассматривается как перспективное направление: **FHE обеспечивает приватность, ZKP – доказательство корректности** [61]. В идеале узел мог бы на своих входных зашифрованных данных посчитать функцию и сразу приложить SNARK, что вычисление проведено правильно. Но на практике пока **совместить FHE и ZKP очень сложно технически** из-за огромных вычислительных затрат и несовместимости некоторых схем шифрования и доказательств [62]. Вероятно, мы увидим прогресс в этом направлении не в ближайшем будущем, но исследования ведутся.

Многопартийные вычисления (MPC). MPC – это крипто-протокол, при котором несколько сторон совместно вычисляют функцию от своих частных входов, не раскрывая их друг другу. Классический пример – несколько больниц хотят обучить общий ML-модель на совокупности своих данных, но не могут обменяться сырыми данными из-за приватности; MPC позволяет им пообщаться через протокол и получить модель, не раскрыв индивидуальные датасеты. MPC можно рассматривать как альтернативу FHE: в FHE вы доверяете схеме шифрования, а в MPC – распределению доверия между несколькими узлами. Если хотя бы один узел честный и не раскроет лишнего, то секреты сохранятся. В контексте децентрализованных вычислений MPC может применяться так: разбить задачу между несколькими узлами, каждый получает зашифрованную/маскированную часть данных и выполняет свою долю, по ходу обмениваясь промежуточной информацией, шифруя её. В итоге они приходят к разделённому результату, который затем объединяется. При этом ни один узел не видит полный объем данных или полный результат, но общий вычислительный труд сделан. Пример – **облачные сервисы шифрования**: можно разбить токен доступа на несколько частей и ни один сервер не будет иметь всего токена целиком, но вместе они смогут выполнить авторизацию. Для ML существуют библиотеки (например, Crypten, TF Encrypted), позволяющие выполнять инференс нейросети в MPC: модель и данные разделяются между узлами, и они выполняют вычисление на зашифрованных шардах, получая тоже зашифрованный итог. Это значит, что *ни модель, ни входы, ни выход* в чистом виде не видны ни одной стороне – важное свойство для доверия. Однако MPC страдает от существенных накладных расходов на коммуникацию: практически на каждую простую операцию узлам нужно обменяться сообщениями. Поэтому, как и ZKP, MPC пока тяжеловесен для больших задач, но уже используется в нишевых сценариях (например, совместное машинное обучение банков с секретными данными клиентов).

Подводя итог, **доверие в децентрализованных вычислениях достигается сочетанием методов**: экономических (стимулы и штрафы), архитектурных (репликация и детерминизм) и криптографических (SNARK-доказательства, FHE, MPC). Каждый из них балансирует компромисс между накладными расходами и уровнем гарантии. В текущих реализациях чаще используют более простые пути (репутация, частичная репликация) [45], но будущее, вероятно, за всё большей ролью криптографии, позволяющей “не доверять, а проверять” любые удалённые расчёты.

Безопасность и ограничения

При переносе вычислений на клиентскую сторону и их распределении по неизвестным узлам возникают вопросы безопасности. С одной стороны, исчезает необходимость хранить пользовательские данные на сервере, что устраняет риск взлома центральной БД – данные либо остаются локально, либо хранятся в зашифрованном виде на децентрализованных узлах. С другой – сам пользовательский окружение может быть более уязвимым (браузер, домашний ПК) по сравнению с защищённым сервером.

Изоляция кода. Одним из преимуществ браузерной архитектуры является песочница: код, выполняемый в браузере, не имеет прямого доступа к файловой системе, произвольным сетевым подключениям или привилегиям ОС [63]. Это повышает безопасность: даже если злоумышленник попытается распространить вредоносный код через децентрализованную сеть, в рамках браузера он будет ограничен. WebAssembly-модули дополнительно изолированы – они не могут выйти за пределы выделенной им памяти и взаимодействуют с внешним миром только через контролируемые вызовы API. В сетях вроде Fluence все пользовательские функции запускаются в виде WASM на узлах-провайдерах, что предотвращает повреждение хоста: код от разных клиентов работает в песочницах и не может получить доступ к данным вне своего окружения [64]. Важным аспектом здесь является *управление побочными эффектами*: Fluence ввела

понятие **Managed Effects** – контролируемые операции, через которые вычислительный модуль может взаимодействовать с внешними системами (будь то обращение к Web2 API или запись в блокчейн) [65]. Узел-исполнитель не выполнит неподписанный или неавторизованный внешний вызов – это предотвращает сценарии, когда удалённый код, например, украдет секрет с узла или пойдёт по сети куда не следует.

Защита узлов. Если пользовательское устройство выделяет часть ресурса для сети (например, отдает немного GPU во флотовое рендеринг), то важно обезопасить его от перегрузки и эксплойтов. На уровне протокола, узлы обычно могут выставлять лимиты: сколько CPU/GPU времени отдавать, какой трафик потреблять. Это защищает от злоупотреблений (нечестный заказчик не сможет бесконечно грузить узел без компенсации). Кроме того, узлы могут отключаться/включаться, и архитектура должна это переносить (Tolerance to churn) – распределённые системы проектируют так, чтобы постоянная доступность отдельного узла не требовалась (репликация состояния, резервирование). От **DDoS-атак** частично спасает сама природа распределённости: нет единого адреса, куда слать терабиты трафика, а нагрузка распределена по множеству машин.

Sybil-атаки – отдельный риск: злоумышленник может запустить множество псевдо-узлов, выдавая их за независимых участников, и попытаться нарушать работу (например, голосуя ложными результатами при репликации вычислений). Защищаются от этого с помощью *экономического барьера*: требование залога или затрат (Proof-of-Work, Proof-of-Stake) для присоединения. Если создание сотни поддельных узлов стоит дорого, атака теряет смысл. Также помогают репутационные системы: каждый узел со временем нарабатывает историю, и новые "однодневки" не будут пользоваться доверием для важных задач.

Передача данных и шифрование. Во всех P2P-коммуникациях должен использоваться **энд-ту-энд шифрование**. В WebRTC это встроено: все соединения шифруются DTLS/SRTP. Для хранилищ вроде IPFS – данные желательно **шифровать на стороне клиента** перед размещением (особенно если это приватные данные), потому что по умолчанию IPFS-контент доступен всем по CID. Многие dApps идут по пути: сохранять в распределённом хранилище только зашифрованные блобы, а ключ хранить у пользователя или распределённо. Таким образом, компрометация какого-либо узла-хранителя или ретривера не раскрывает пользовательскую информацию (без ключа данные бессмысленны). Для обмена сообщениями между браузерами применяют проверенные крипто-библиотеки (например, Signal Protocol для чат-приложений) – это обеспечивает конфиденциальность общения даже без собственного сервера.

Вопросы обновлений и доверия к коду. Интересно, что в полностью клиентской модели обновление приложения означает просто обновление фронтенд-файлов. Пользователи всегда получают последнюю версию при перезагрузке страницы (или PWA) [66]. Но здесь скрыт нюанс безопасности: если разработчик выпустил злонамеренную обнову фронтенда или его ключ для подписи обновления украдут, все пользователи автоматически загрузят новый код. Без серверной валидации у пользователя мало способов защититься, кроме как полагаться на **аудит смарт-контрактов или открытый исходный код фронта**. В идеале, эволюция должна прийти к тому, что и сам код фронтенда может храниться и распространяться децентрализованно (например, в IPFS+ENS), а сообщество может отслеживать изменения (через хеши версий).

В целом, при правильном подходе, **безопасность распределённых приложений может превосходить традиционные**: нет концентрации данных для крупных утечек, нет центрального сервера для компрометации. Но ответственность за определённые аспекты (шифрование данных, управление ключами) больше ложится на пользователя и разработчика приложения. Требуется продуманная криптографическая архитектура, чтобы пользователю не пришлось

"доверять" скрытому бэкенду – ведь его и нет, а вся логика защиты должна быть явно реализована на клиентской стороне.

Производительность и масштабируемость

Децентрализованные вычислительные системы обещают практически неограниченную масштабируемость за счёт привлечения глобального пула ресурсов [5]. Однако на практике их производительность зависит от множества факторов: пропускной способности сетей между узлами, накладных расходов на координацию и проверку, качества соединений и оборудования на узлах-участниках.

Локальная производительность vs удалённая. Выполнение кода на самом клиенте (особенно с WebGPU) иногда может быть **быстрее**, чем отправка запроса на сервер и ожидание ответа. Это устраняет сетевую задержку – важный фактор для интерактивных приложений. Например, рендеринг графики в браузере через WebGPU происходит без участия удалённого API, поэтому реакция мгновенная. Аналогично, сортировка большого массива данных в браузере может быть быстрее полного цикла запрос-ответ к серверу за теми же данными. Современные устройства способны взять на себя заметную долю работы, разгружая центральные серверы [18]. Но если задача чрезвычайно тяжёлая (долгая по времени или требующая специализированного железа), то один клиент может не справиться или будет очень медленным. Тогда вступает распределение между узлами сети.

Масштабирование горизонтальное. В распределённой сети можно увеличивать общую вычислительную мощность практически линейно с добавлением новых узлов – так, подключив 100 узлов, можно теоретически считать задачи в 100 раз быстрее, чем на одном (при идеальном параллелизме). Это особенно полезно для *параллельно разложимых* задач: рендеринг различных кадров анимации, применение одного алгоритма к множеству независимых данных (например, рендеринг множества изображений, пакетная обработка фотографий), распределённый перебор или эволюционные алгоритмы. Здесь децентрализация проявляет себя во всей красе – можно задействовать тысячи машин по всему миру. Более того, такие сети не имеют *единого узкого места*: если один узел выходит из строя, остальные продолжают, а потерянная часть работы переназначается. Это повышает **устойчивость** к сбоям оборудования.

Накладные расходы и задержки. Тем не менее, распределённые вычисления сталкиваются с задержками распространения данных и результатом. Если задача требует постоянного обмена большими промежуточными данными между узлами (например, при обучении нейросети каждый шаг требует синхронизации градиентов между рабочих узлов), то узким местом может стать сеть. *Пропускная способность интернета* у разных узлов разная, и суммарно кластер на одной локальной станции может обучить модель быстрее, чем такое же число узлов разбросанных по миру, из-за задержек связи. Протоколы типа Gensyn NoLoCo пытаются минимизировать эти синхронизации [67], но физику не обмануть: географически распределённая сеть всегда будет иметь бóльшие латентности. Поэтому *не все типы задач эффективно распределяются* глобально – некоторые лучше выполнять "на краю", ближе к источнику данных. Например, **концепция "data gravity"** гласит: большие объёмы данных притягивают вычисления [25]. И Filecoin-инфраструктура следует ей, перемещая вычислительные задачи туда, где хранятся данные, чтобы не гнать петабайты по сети [26].

Параллелизм vs последовательность. Децентрализация наиболее продуктивна для параллельных задач. Если же задача строго последовательна и не может быть разделена, распределённая сеть её не ускорит (она всё равно должна по сути выполняться на одном узле

или на каждом узле последовательно). Однако можно все равно получить выигрыш, например, выполнив несколько *независимых последовательных задач* на разных узлах параллельно. В целом, для достижения высокой производительности приложения приходится разрабатывать с учётом параллелизма и **асинхронности**. Здесь веб-технологии тоже эволюционируют: в браузере есть Web Workers для многопоточной работы, SharedArrayBuffer для общих данных между потоками – т.е. даже на клиенте можно использовать все ядра CPU. А по сети – асинхронные запросы по P2P могут выполняться конкурентно, и потом объединяться.

Надёжность и повторное выполнение. В распределённой среде часто узлы могут отвалиться в любой момент. Протоколы должны предусматривать повтор задания на другом узле, если первый не ответил за оговорённое время. Это добавляет непредсказуемость в время выполнения задачи. Эффективность зависит от доли ненадёжных узлов: если 90% узлов исполняют быстро и верно, а 10% выпали – то будет небольшой оверхед; если же половина узлов постоянно нестабильна, система будет тратить много времени на перераспределение. Для смягчения этого применяют избыточное назначение: могут сразу задать задачу двум узлам параллельно и принять первый ответ, второй просто отбросив. Такой *спекулятивный запуск* увеличивает шанс получить результат быстрее, но тратит лишние ресурсы.

Сравнение с централизованным облаком. В традиционном облаке (AWS/Azure) латентности минимизированы: серверы в одном датацентре соединены скоростными магистралями, ресурсы однородны, задачи распределяются оптимально. Децентрализованная сеть пока не может полностью повторить этот уровень оптимизации: узлы разнородны (разные CPU/GPU, разная скорость), сеть между ними медленнее и менее стабильна. Поэтому по *сырым показателям* (флопс в секунду на задачу) распределённые сети могут уступать хорошо настроенному централизованному кластеру. Однако **экономический и географический масштаб** децентрализации означает, что можно привлечь больше суммарных ресурсов дешевле. Например, Render Network заявляет, что имеет больше GPUs совокупно, чем крупные облачные провайдеры, и по существенно меньшей цене за час [68] [69]. Это возможно за счёт того, что используются простаивающие потребительские GPU по всему миру, которые иначе вообще не работали бы. В итоге заказчик за ту же сумму получает больше GPU-часов, хоть и с чуть меньшей надёжностью.

Эволюция производительности. Можно ожидать, что с развитием технологий наподобие **децентрализованных сетей доставки контента (CDN)**, узлы для вычислений тоже станут более *близкими к пользователю*. Появится больше *edge*-пиров (на уровне интернет-провайдеров, городских узлов), предлагающих свои мощности с малой задержкой для локальных пользователей. Тогда разница между "своим сервером в соседнем городе" и "децентрализованным провайдером в том же городе" сойдёт на нет. Кроме того, совершенствование алгоритмов (как распределённое обучение без глобальной синхронизации) будет сокращать потери эффективности. Как отмечалось, решения вроде **параллельного конвейера SkipPipe** уже позволяют продолжать обучение модели даже при одновременном сбое половины узлов, лишь с небольшим деградированием скорости [70]. Всё это призвано сделать распределённые системы более *производительными и устойчивыми*, приближая их по удобству к привычным облачным сервисам.

Применение и эволюция экосистемы Web3 без централизованных серверов

Рассмотрим, как на практике может выглядеть приложение, полностью работающее без собственного сервера, и какие технологии Web3 оно объединяет. В такой **продуктовой**

архитектуре фронтенд (написанный на TypeScript, работающий в браузере или как мобильное/настольное приложение через webview) взаимодействует с множеством децентрализованных сервисов:

- **Данные и файлы**: загружаются и сохраняются через децентрализованные хранилища. Статический контент приложения (HTML/JS/CSS, медиаресурсы) может размещаться в IPFS или на блокчейне (например, в сети Arweave) – пользовательский клиент скачивает их оттуда, без центрального веб-сервера. Пользовательские файлы (фото, документы) также уходят в IPFS/Filecoin, но предварительно шифруются на клиенте для приватности. В итоге ни одна центральная организация не контролирует хранение – файлы **реплицируются на множестве узлов**, что предотвращает цензуру и потери данных.

- **Идентификация и права**: реализуются через криптографические кошельки и децентрализованные идентификаторы. Пользователь заходит в приложение, используя, например, Ethereum-кошелёк или DID, который хранится у него. Любые действия подписываются его приватным ключом. Вместо традиционной базы данных пользователей и сессий, приложение проверяет подписи – так достигается **авторизация без сервера**. Права доступа к данным могут контролироваться схемами вроде NFT-токенов или записей в смарт-контрактах (например, только владелец определённого токена может расшифровать соответствующий файл).

- **Бизнес-логика**: ключевая логика, связанная с состоянием, выносится в **смарт-контракты** на блокчейне или в другие проверяемые среды. Например, если приложение – это торговая платформа, то сделки и балансы пользователей хранятся в смарт-контракте (децентрализованной и неизменяемой программе) – таким образом, не нужен бэкенд-сервер, выступающий бухгалтером. Фронтенд отправляет транзакции в сеть блокчейна, где они выполняются, и читает состояние (по защищенному API типа Infura или через собственный light node). Это гарантирует **прозрачность и неизменность**: участники доверяют коду контракта, а не компании-разработчику. Примеры: децентрализованные биржи (Uniswap) – их фронтенд это просто интерфейс к контрактам Ethereum, обмен проходит полностью на блокчейне без серверов [71] . Коллекционные игры (CryptoKitties) – вся логика владения и обмена активами также в блокчейне, фронтенд только вызывает её [72] .

- **Вычисления и обработка**: тяжёлые расчёты, которые нецелесообразно делать в блокчейне (например, генерация рекомендаций, тренировка моделей, рендеринг видео по запросу), выполняются через подключение к **распределённым вычислительным сетям**. Например, приложение видеомонтажа может на клиенте собрать проект, а финальный рендер ролика поручить сети типа Render Network – она распределит работу по узлам GPU и вернёт готовое видеофайл, сохранённый в IPFS. При этом заказ рендеринга и оплата могут быть оформлены смарт-контрактом, а результат (файл) обеспечен доказательством сохранности (через контент-хеш IPFS). Другой пример – приложение аналитики больших данных: фронтенд формирует запрос, через Filecoin Bacalhau отправляет задачу “обработать dataset X таким-то образом” – узлы выполняют и заносят обратно результат в сеть, откуда фронтенд скачивает его и визуализирует.

- **Реалтайм связь и кооперация**: для функций вроде чата, совместного редактирования или игр, используются P2P-сети или федеративные протоколы. Например, мессенджер может работать поверх протокола Matrix или использовать децентрализованный pub/sub. Клиенты обменятся сообщениями напрямую или через распределённые ретрансляторы, но **ни одного центрального сервера, хранящего всю переписку**, нет. Сообщения

шифруются end-to-end, адресуются через децентрализованные ID. Подобные концепты (например, **Secure Scuttlebutt** – социальная сеть P2P) показывают, что даже соцсеть или форум можно построить без серверного бэкенда: посты распространяются между узлами, хранятся локально и у подписчиков, а не на сервере [73].

Такая архитектура, конечно, более **сложна** в разработке сегодня, чем классическая. Требуется интегрировать несколько распределённых компонентов, обеспечить их согласованную работу и хороший UX (что тоже вызов – например, заставить пользователя иметь крипто-кошелёк или ждать подтверждения транзакций). Однако она приносит ценность: **отпадает необходимость доверять одну точку**, всё построено на открытых протоколах. Пользователь ощущает, что приложение *принадлежит ему*, его данные под контролем, а сервис доступен, пока жив хотя бы сам пользовательский софт и распределённые узлы.

Эволюция экосистемы. Можно предположить, что со временем появятся стандартные "строительные блоки" для таких приложений. Уже сейчас многие используют комбо: хранение через IPFS/Filecoin, аутентификация через Ethereum (MetaMask), бэкенд-логика через контракты. С появлением децентрализованных вычислительных сетей, следующий шаг – заменить централизованные API для сложных функций. Если раньше даже в dApp приходилось где-то на сервере делать, к примеру, ML-инференс (потому что на блокчейне это невыполнимо, а на фронте слишком тяжело), то вскоре можно будет звать, например, **децентрализованное API для AI**. Проекты вроде **io.net** и **Gensyn** по сути предлагают *ML-aaS без конкретного владельца*: разработчик платит сети токенами, а сеть возвращает ему результат, как обычный облачный сервис, но в бэкэнде которого – тысячи компьютеров по всему миру вместо одного датацентра [74].

Важно, что акцент смещается **с финансового аспекта Web3 на прикладной**. Пользователя не обязательно будет волновать, какой токен под капотом сети обеспечивает её работу – идеальный сценарий, когда они взаимодействуют с децентрализованной системой так же просто, как с обычным веб-сервисом, просто получая больше гарантий и прозрачности. Например, в браузере приложение вызывает функцию распознавания речи, а под капотом происходит распределённый запрос к GPU-сети, оплата через микротранзакцию Lightning или крипто, – всё это скрыто и происходит за секунды. Пользователь лишь видит, что сервис сработал и приватность его соблюдена.

Экосистема движется к **убиранию центральных API и серверов** там, где это оправдано. Уже есть полностью децентрализованные обменники, хранилища, именованные DNS (ENS), и постепенно появляется вычислительный слой. В обозримом будущем разработчики смогут собирать приложения полностью из этих блоков: например, *P2P-хостинг* + *блокчейн-логика* + *децентр.вычисления* + *P2P-коммуникация*, не поднимая ни одного собственного сервера. Это не значит отсутствие затрат – ресурсы оплачиваются, но напрямую владельцам узлов, а не монополистам. И это не панацея для всех случаев – где-то централизованный подход всё ещё проще и эффективнее. Однако в сферах, где критичны открытость, **устойчивость к цензуре** и владение сообществом, такой архитектурный сдвиг очень значим.

Уже сейчас команды по всему миру экспериментируют с подобными продуктами – от анонимных мессенджеров на блокчейне [75] до офисных приложений, работающих локально (принцип *local-first*, когда синхронизация – опция, а не требование). Децентрализованные вычислительные сети только начинают внедряться, но обещают снять последние барьеры – выполняя на открытой инфраструктуре даже сложные задачи, которые раньше вынужденно доверяли централизованным сервисам (например, анализ больших данных или AI). **Будущее экосистемы Web3** видится как **кооперативное облако**, где приложения принадлежат своим пользователям, а

ресурсы предоставляются множеством добровольцев или провайдеров по общим правилам. Это расширит спектр возможных приложений и снизит входной порог для стартапов (им не нужно будет выстраивать дорогостоящий бэкенд, достаточно задействовать готовые децентрализованные сервисы). В свою очередь, пользователи получат больше контроля и уверенности: "прозрачность и открытость данных, неизменность логики" гарантируются протоколом, а не обещаниями компании [76]. Конечно, ещё предстоит работа над юзабилити и скоростью, но тенденция очевидна – **интернет-сервисы становятся более распределёнными и доверенными**, возвращаясь к изначальным принципам сети, где "никто не владеет всем". Это большой шаг к открытой, устойчивой и демократичной вычислительной экосистеме [77].

Заключение

Децентрализованные вычислительные системы, опирающиеся на клиентские устройства и P2P-инфраструктуру, представляют новый виток в архитектуре приложений. Благодаря WebAssembly и WebGPU, современные клиенты способны выполнять сложные алгоритмы и модели локально, снижая зависимость от серверов. А глобальные распределённые сети – от хранилищ Filecoin до GPU-рынков типа Render или Gensyn – позволяют вынести оставшиеся тяжёлые задачи в **безсерверное облако**, где ни один игрок не контролирует вычислительные ресурсы. В таких системах повышается устойчивость (нет единой точки отказа), масштабируемость достигается горизонтально, а приватность пользовательских данных улучшается, так как данные могут не покидать устройство или шифроваться на всём пути [8].

Однако, вместе с преимуществами приходят и технические вызовы: как убедиться в честности удалённых результатов (решаются через репликации, криптодоказательства, стимулы) [45] [57], как организовать удобную координацию без центрального узла (новые протоколы и языки вроде Fluence Aqua решают этот вопрос) [41], как обеспечить сопоставимую с облаком производительность (требует умных алгоритмов распределения и учета факторов “где данные – там и вычисления” [26]).

Текущие проекты уже показывают жизнеспособность подхода: в Filecoin-сообществе активно развиваются решения для децентрализованных вычислений рядом с данными [78], платформы типа Fluence предлагают разработчикам инфраструктуру для **peer-to-peer функций** и снижения затрат на 85% по сравнению с традиционным облаком [64]. Большие инвестиции (например, $43M в Gensyn) говорят о вере в то, что **демократизация AI через открытые вычислительные протоколы** возможна и востребована.

Таким образом, мы наблюдаем формирование полноценного **технологического стека Web3**, где фронтенд не просто интерфейс, а главный исполнитель, взаимодействующий с распределёнными сетевыми “сервисами”. Постепенно становятся реальностью приложения, которые *не имеют скрытого бэкенда*: весь их “бэкенд” – это блокчейн, IPFS, децентрализованные вычислительные узлы и прямое P2P-взаимодействие. Это значимое изменение парадигмы, которое может сделать интернет более **устойчивым, открытым и подконтрольным пользователям**. Разумеется, не все приложения перейдут на такую модель – она сложнее, и порой избыточна. Но в тех областях, где доверие и автономность на первом месте (финансы, коммуникации, управление данными, AI), децентрализованная архитектура обещает большие выгоды. Мы стоим в начале этого пути, и предстоящие стандарты, проекты и общие усилия определят, насколько массовой и удобной станет эпоха **безсерверных, децентрализованных вычислений**. Как показывает опыт, технология движется быстро: возможно, уже в ближайшие годы мы увидим всё больше приложений, где о сервере не вспоминает никто, кроме историков.

Источники: Использованы материалы и примеры из текущих проектов и исследований децентрализованных вычислений, включая Filecoin/Bacalhau [29] [51], Render Network [31], Gensyn [36], Fluence [41], а также обзоры по ZK-ML и доверенным вычислениям [45] [57] для освещения соответствующих технических аспектов.

---

[1] [5] [25] [26] [29] [40] [42] [43] [50] [51] [74] [77] [78] Unleashing the Power of Decentralized Compute with Filecoin | Filecoin Foundation
https://fil.org/blog/unleashing-the-power-of-decentralized-compute-with-filecoin

[2] [3] [4] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [63] [66] [71] [72] [73] [75] [76] Веб-разработка только на фронтенде_ анонимность, шифрование, P2P и блокчейн.pdf
file://file_00000000a07871f4a28bb16c18e2abe3

[27] [28] Bacalhau: Fast, Secure, and Cost-Efficient Computation | Filecoin Foundation
https://fil.org/ecosystem-explorer/bacalhau

[30] What Is Render Network and How It Rents Out GPU Power
https://www.coingecko.com/learn/what-is-render-network-rndr-crypto

[31] [32] Render Network Knowledge Base
https://know.rendernetwork.com/

[33] Stability AI, OTOY, Endeavor, and The Render Network Join Forces ...
https://www.prnewswire.com/news-releases/stability-ai-otoy-endeavor-and-the-render-network-join-forces-to-develop-next-generation-ai-models-ip-rights-systems-and-open-standards-powered-by-decentralized-gpu-computing-302091818.html

[34] [35] [36] [37] [38] [39] [54] [55] [67] [70] Gensyn | Research
https://www.gensyn.ai/research

[41] [44] [47] [48] [49] [64] [65] Understanding Fluence: Introduction to Cloudless Concepts - Fluence
https://www.fluence.network/blog/understanding-fluence-introduction-to-cloudless-concepts/

[45] [46] [57] [58] [59] [60] [61] [62] Trustless and Decentralized Machine Learning with Zero-Knowledge Proofs and Blockchains
https://www.cvvc.com/blogs/trustless-and-decentralized-machine-learning-with-zero-knowledge-proofs-and-blockchains

[52] [53] [68] [69] io.net | Decentralized GPU Cloud
http://io.net

[56] ZKML: Verifiable Machine Learning using Zero-Knowledge Proof
https://kudelskisecurity.com/modern-ciso-blog/zkml-verifiable-machine-learning-using-zero-knowledge-proof