DOI: 10.58168/PPSPM2025_108-120
УДК:004.9

БАЗЫ ДАННЫХ ДЛЯ ГЕНЕРАТИВНЫХ МОДЕЛЕЙ: АРХИТЕКТУРЫ, ВЫЗОВЫ И ПЕРСПЕКТИВЫ

В.И. Куницын, В.С. Лучников, Е.С. Ильин, А.О. Карташов

ФГБОУ ВО «Воронежский государственный лесотехнический университет имени Г.Ф. Морозова»

Аннотация. Развитие генеративного искусственного интеллекта — от крупных языковых моделей до диффузионных и состязательных архитектур — обнажило стратегическую роль данных. Модели создают новый контент, но качество результата определяется тем, как устроены процессы сбора, хранения, индексирования и доставки обучающих и вспомогательных наборов. Классические реляционные СУБД и распространённые NoSQL- хранилища предложили основу для многих проектов, однако специфика генеративных систем сместила фокус: возникает потребность в хранении векторных представлений (embeddings), в поиске по смысловой близости, в высокой скорости выборок. Эта монография систематизирует архитектуры баз данных, применимых в генеративных сценариях, рассматривает методы индексирования и интеграции с моделями, а также анализирует риски и направления дальнейшего развития.

Ключевые слова: генеративные модели; базы данных; векторные базы данных; embeddings; семантический поиск; RAG; индексирование; машинное обучение; безопасность данных; федеративное обучение.

DATABASES FOR GENERATIVE MODELS: ARCHITECTURES, CHALLENGES, AND PROSPECTS

V.I. Kunitsyn, V.S. Luchnikov, E.S. Ilyin, A.O. Kartashov

Voronezh State University of Forestry and Technologies named after G.F. Morozov

Abstract. The rise of generative artificial intelligence—from large language models to diffusion and adversarial architectures—has laid bare the strategic role of data. Models create new

content, but the quality of their outputs is determined by how training and auxiliary datasets are collected, stored, indexed, and delivered. Traditional relational DBMSs and widely used NoSQL stores have provided the foundation for many projects; however, the specifics of generative systems have shifted the focus: there is a need to store vector representations (embeddings), to perform similarity-based (semantic) retrieval, and to support high-throughput reads during inference. This monograph systematizes database architectures applicable to generative scenarios, reviews indexing methods and integration with models, and analyzes risks and future directions.

Keywords: generative models; databases; vector databases; embeddings; semantic search; RAG; indexing; machine learning; data security; federated learning.

Генеративные модели и требования к данным

Генеративные модели ориентированы не на классификацию, а на синтез: они конструируют новый текст, изображение или иные виды данных, опираясь на статистические закономерности, выявленные в больших корпусах. Выделяют несколько доминирующих семейств. Большие языковые модели (LLM) обучаются на многожанровых текстовых массивах и используют вероятностные распределения для предсказания следующего токена; от качества и разнообразия корпуса зависит их способность к обобщению и переносимости между предметными областями. Генеративно- состязательные сети (GAN) используют противостояние генератора и дискриминатора, что заставляет модель производить всё более правдоподобные образцы. Диффузионные архитектуры, наоборот, стартуют с шума и в серии шагов восстанавливают структуру данных, что даёт высокую детализацию изображений и устойчивость к артефактам.
Роль базы данных проявляется уже на этапе конструирования датасета: необходимо поддерживать поступление потоков из разнородных источников, исключать дубликаты, фиксировать происхождение (datalineage) и версионировать выборки. Во время обучения требуется устойчивый throughput при чтении больших последовательных блоков, а в инференсе — быстрый доступ к вспомогательным сведениям: справочникам, индексам, кэшам и, главное, к векторным представлениям. Отдельная задача — актуальность: языковые модели без регулярной подкачки свежего материала быстро теряют связь с реальными событиями. Наконец, критично качество: присутствие токсичных, разметочно ошибочных или юридически спорных данных воспроизводит и усиливает искажения в ответах модели.

В практических проектах генеративного ИИ вопросы «каких данных много» и «какие именно данные нужны» никогда не совпадают. Объём сам по себе не спасает: модель чувствительна к распределению источников, к повторяемости сюжетов, к свежести и юридической чистоте корпуса. Поэтому контур работы с данными строят как управляемый процесс, а не набор разрозненных загрузок. Сначала определяют продуктовую цель (диалоговые сценарии, код-ассистент, визуальный генератор), затем описывают целевые домены и фиксируют квоты — сколько материалов из каждого домена должно попасть в выборку. Этот шаг снижает риск «перекоса» модели в сторону избыточно представленного класса документов.

Далее — этап «санации» источников. Для текста это удаление boilerplate, исправление явных кодировочных артефактов, нормализация типографики. Для изображений — проверка разрешения, фильтр по форматам, отбрасывание пустых или повреждённых файлов. Для аудио и видео — проверка длительности, корректный контейнер, при необходимости — автоматическая транскрипция с маркировкой качества. На этом же шаге по возможности устраняют юридические риски: отсеивают ресурсы с неясной лицензией, фиксируют происхождение и добавляют «паспорт» набора (datasetcard) с описанием допущений и ограничений.

Отдельной строкой идёт дедупликация и борьба с «почти дублями». Хэш-подписи (SimHash, MinHash) обнаруживают повторяющиеся куски текста; для изображений используют перцептивные хэши и алгоритмы поиска ближайших соседей. На больших объёмах важен компромисс: слегка агрессивная дедупликация лучше, чем накачивание модели бесконечными повторами, которые искусственно сужают разнообразие. Параллельно запускают поиск персональных данных: регулярные выражения и NER-модели находят потенциальные PII-сущности, после чего включается политика редактирования (редакция, маскировка, а в спорных случаях — исключение документа).

Когда «сырьё» приведено в приличный вид, начинается курирование и аннотирование. В генеративных задачах редко хватает полностью ручной разметки — её дополняют слабым обучением (weaksupervision), активным обучением (activelearning) и самопроверками модели. Для LLM это ещё и выбор стратегии токенизации, «нарезка» на фрагменты (semanticchunking против скользящего окна), а также контроль доменных квот: если целевая область — медицина или право, общий корпус должен быть смешан с профильными наборами так, чтобы сигнал из домена не потерялся в общем шуме.

Финальный слой — версионирование и защита от утечек смысла. Наборы публикуют как vX.Y с неизменяемыми чек-суммами и описанием изменений; отдельным прогоном проверяют «контаминацию» контрольных задач (eval), чтобы тесты не оказались в обучении. Только после этого данные собирают в шардированные хранилища для обучения и раскладывают по векторным витринам для инференса: embeddings-хранилища, кэши RAG, справочники и словари.

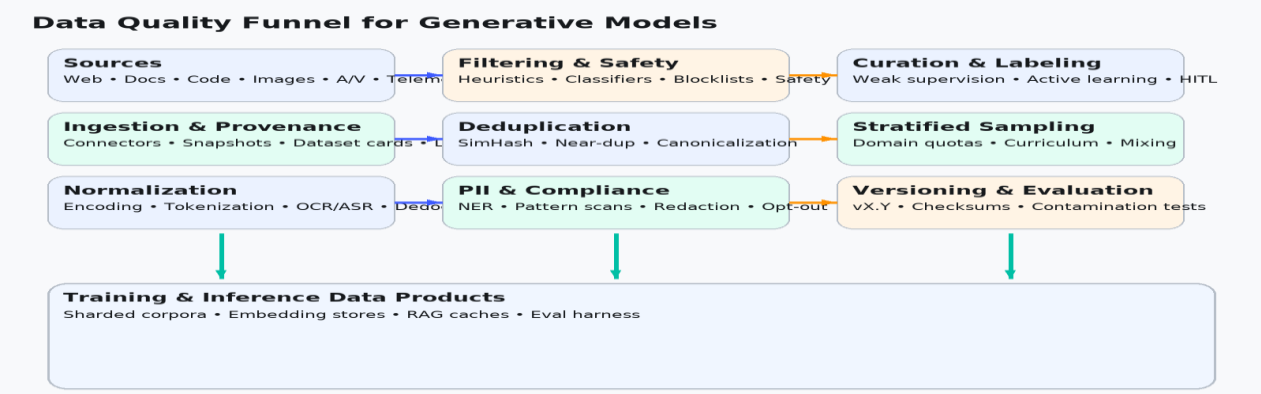

Рисунок 1 - Воронка качества данных для генеративных моделей

Диаграмма построена как трёхколоночная «воронка». Слева — источники и приём данных: веб-корпуса, документы, кодовые репозитории, изображения/аудио, телеметрия. Здесь же фиксируются происхождение и лицензии, формируются datasetcards. В средней колонке показаны блоки качества и соответствия требованиям: фильтрация нежелательного контента, автоматические классификаторы безопасности, дедупликация (включая поиск «почти дублей»), удаление PII с помощью NER и регулярных шаблонов, а также учёт согласий/отказов. Справа — курация и выборки: слабая и активная разметка, включение человека в цикл, стратифицированная выборка с доменными квотами и элементами «учебной программы» (curriculum), затем — версионирование наборов (vX.Y), контроль целостности и отдельные прогоны на предмет контаминации тестов.

Все три «рукава» сходятся в нижний блок: готовые продукты данных для обучения и инференса — шардированные корпуса, векторные витрины (embedding-store) для семантического поиска и RAG-кэши, а также инфраструктура оценивания (evalharness). Такая организация позволяет не просто «набить» модель данными, а поддерживать управляемое качество, отслеживаемость изменений и воспроизводимость экспериментов. Это особенно важно, когда модель разворачивается в промышленной среде и

требования к правовой чистоте и безопасности не уступают требованиям к точности.

Архитектуры баз данных для генеративных систем

Под генеративные сценарии складываются четыре базовых класса хранилищ. Реляционные СУБД обеспечивают строгую схему, транзакции и зрелые механизмы целостности; они незаменимы для учёта, метаданных, прав доступа и отчётности, но не оптимальны для семантического поиска. NoSQLсемейство (документные, колоночные, keyvalue и widecolumn решения) берёт на себя масштабируемость и гибкую схему, что полезно для хранения сырья корпусов и событий. Векторные базы данных адресуют главный «бутылочный горлышок» генеративных систем — быстрый поиск по embedding векторам и топ K ближайших соседей. Наконец, гибридные платформы совмещают перечисленные подходы: векторные индексы соседствуют с транзакционной зоной и аналитическим слоем.

Таблица 1 – Сравнение SQL, NoSQL и векторных БД

Аспект	SQL (реляционные)	NoSQL	Векторные БД
Модельданных	Строгаясхема, нормализация	Гибкая/схема-он-рид	Векторы + метаданные
Транзакции	ACID, зрелыемеханизмы	Ограничено/BASE	Часто eventual + транзакции по метаданным
Масштабирование	Вертикально + шардинг	Горизонтально из коробки	Горизонтально, шардированиепосегментам
Поиск	Ключи/диапазоны	Гибкиепаттерны	ANN/косинус/евклид, топ- K
Скоростьзаписи	Стабильная, транзакционная	Высокаядляпотоков	Зависитотиндекса/батчей
Интеграции	BI/ETL, аудит	Стриминг, IoT, события	RAG, LLM- инференс
Сценарии	Учёт, метаданные, права	Корпуса, логи, телеметрия	Embeddings, семантическийпоиск
Стоимость	Зрелыеинструменты, DBA	Сложностькластеров	Индексы и хранение векторов дороже
Сильныестороны	Целостность, отчётность	Гибкость, масштаб	Смысловойпоиск, rerank
Ограничения	Нетсемантическогопоиска	Нетстрогойтранзакционности	Сложностьтюнингаиндексов

Сравнение показывает, что реляционные хранилища остаются опорой для критических метаданных и аудита; NoSQL- подходы покрывают потребность в приёме и хранении неструктурированных корпусов; векторные базы обеспечивают семантический поиск и RAG- сценарии. На практике эти классы комбинируют: транзакционный контур живёт в SQL, сырьё — в документном/объектном слое, а поверх размещается векторный индекс с быстрым ANN- поиском.

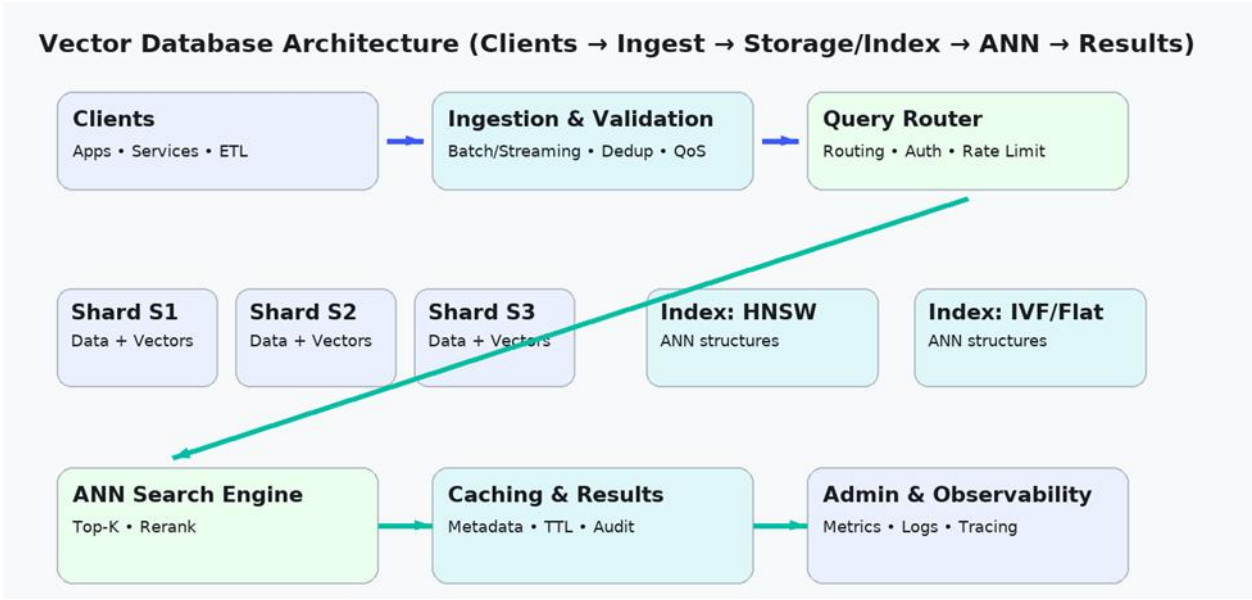

Рисунок 2 - Архитектура векторной базы данных

Диаграмма отражает сквозной путь данных: клиенты генерируют запросы и потоки записи, конвейер приёма валидирует и дедуплицирует события, маршрутизатор распределяет нагрузку и выполняет аутентификацию. Шардированное хранилище держит векторы и метаданные; индексы (HNSW/IVF) обслуживают приближённый поиск. Движок ANN формирует топ- K соседей, кэш сохраняет частые результаты, а слой наблюдаемости собирает метрики и трассировки.

Методы хранения и индексирования данных

Векторные представления стали «рабочей единицей» генеративных систем. Для эффективного доступа используют структуры и алгоритмы приближённого поиска ближайших соседей (ANN): графовые индексы HNSW дают высокую точность при умеренной латентности; разбиение пространства на ячейки (IVF) ускоряет отбор кандидатов; квантование (PQ/OPQ) снижает объём хранения и повышает пропускную способность. На практике методы

комбинируют: быстрый грубый отбор по IVF/PQ, затем точный пересчёт расстояний и rerank. Системные аспекты включают батч- вставки, хранение «холодных» векторов в сжатии и GPU- ускорение.

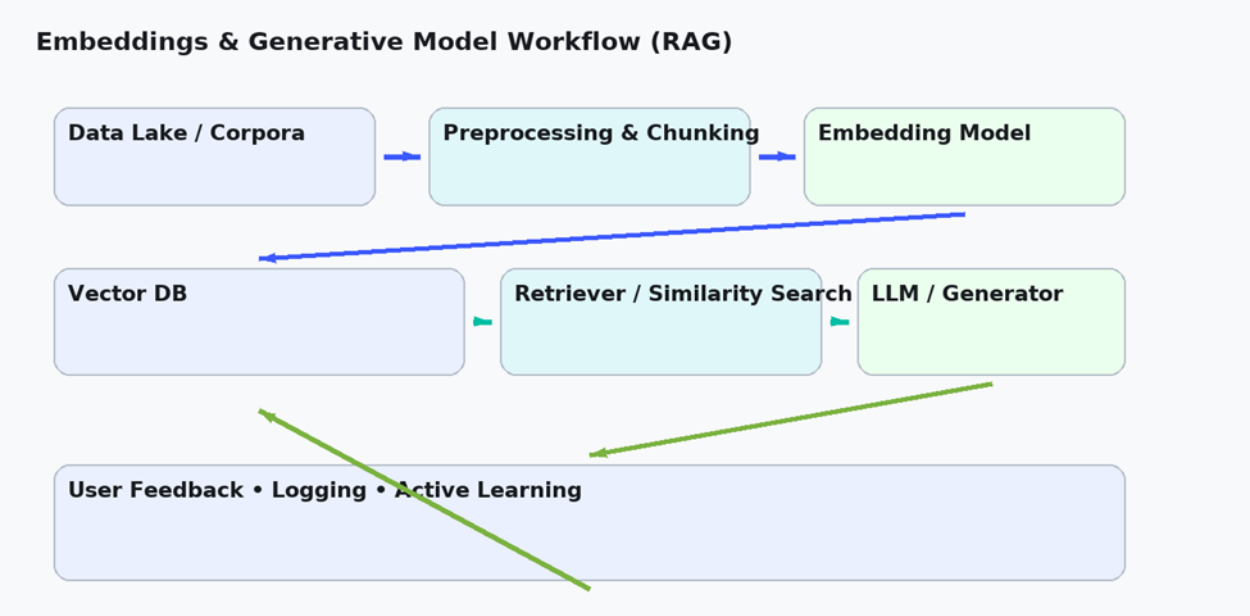

Рисунок 3 - Цикл работы с embeddings в генеративной системе (RAG)

Это схема типичного контура RAG (Retrieval-Augmented Generation), где вокруг генеративной модели выстроен «пояс» из данных и поисковых механизмов.

Верхний ряд отражает офлайн-поток подготовки корпуса. Слева аккумулируются источники (datalake: документы, код, страницы, медиа). Далее идёт санитария и «нарезка» — текст нормализуют, убирают мусор, разбивают на фрагменты удобного размера, чтобы их потом можно было адресовать по смыслу. На следующем шаге специальная модель кодирует каждый фрагмент в числовой вектор. Эти векторы — компактные смысловые отпечатки — и станут основой для поиска. Длинная синяя стрелка намекает, что после кодирования данные попадают в векторное хранилище и этот процесс периодически повторяется пакетами: новые документы доезжают, старые переиндексируются.

Средний ряд — это уже онлайн-инференс. Приходит запрос пользователя; модуль-ретривер обращается к векторной БД и по близости векторов быстро находит несколько наиболее подходящих фрагментов (обычно топ-K). Возможна фильтрация по метаданным: язык, дата, доступ. Отобранные кусочки передаются генеративной модели в качестве контекста, и та формирует ответ — не «из головы», а опираясь на найденные материалы. За счёт этого ответы

становятся актуальнее и проверяемее, а сама модель может быть компактнее, потому что часть «знаний» хранится снаружи.

Нижняя полоса отвечает за контур улучшений. Здесь собираются логи, оценки пользователей, сигналы активного обучения. Зелёные стрелки показывают два канала обратной связи: один возвращается к ретриверу (настраиваются пороги, переранжирование, словари синонимов), второй — к векторной базе (добавление или исправление документов, повторное кодирование, переиндексация). Если пользователи часто правят ответы по определённой теме, эти примеры переходят в «тяжёлые» кейсы для тонкой настройки или дообучения модели.

Рисунок 3 не просто перечисляет блоки, а показывает живой цикл: корпус поступает и чистится, фрагменты кодируются и индексируются, на запросах выполняется быстрый смысловой поиск, генератор собирает итог и отправляет его пользователю, а наблюдение за качеством возвращает сигналы назад, чтобы подстроить ранжирование и обновить базу. Именно эта петля — «подготовка → поиск → генерация → обратная связь → обновление» — и делает RAG-систему устойчивой к изменчивым данным и требованиям.

Интеграция баз данных и генеративных моделей

Интеграция строится вокруг обучения, инференса и эксплуатации. В обучении важны стабильная подача данных и воспроизводимость; в инференсе — минимальная задержка семантического поиска; в эксплуатации — аудит, трекинг качества ответов и управление версиями корпусов. В облачных сценариях используют многорегиональные развертывания, кросс‑региональную репликацию и периметральные кэши (CDN).

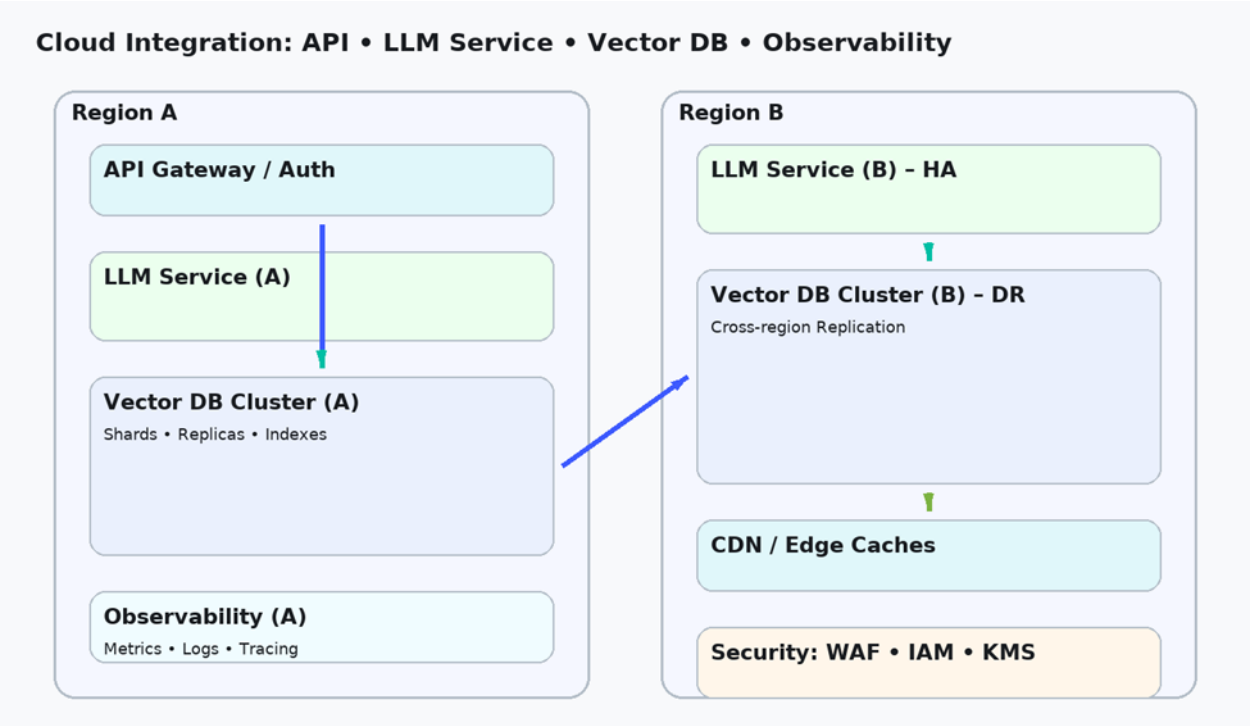

Рисунок 4 – Интеграция базы данных и LLM в облачной среде

На схеме показана «бóевая» раскладка сервиса с генеративной моделью в облаке, разнесённая по двум регионам. Левый блок — основной регион A, правый — резервный регион B. Идея в том, чтобы весь путь запроса — от входа пользователя до чтения из векторного индекса и обратно — работал даже при частичных отказах и был прозрачно наблюдаем.

Обычный сценарий выглядит так. Запрос приходит в регион A через шлюз API: здесь проходят аутентификация, лимиты, базовые проверки. Дальше управление получает сервис LLM. Когда модели нужен фактический контекст (справки, документы, ответы предыдущих пользователей), она обращается вниз — к кластеру векторной БД этого же региона. Отсюда и вертикальная стрелка: генератор не тянет «знания» изнутри, а подтягивает релевантные фрагменты по семантической близости из внешнего индекса. Кластер хранит шардированный массив векторов с репликами и индексами ANN; это позволяет выдерживать и скорость, и объём.

Правый столбец — «зеркало» первой площадки, но со сдвигом по ролям: здесь сервис LLM помечен как высокодоступный, а кластер векторной БД — как контур аварийного восстановления. Косая синяя стрелка между кластерами означает кросс-региональную репликацию: изменения индексов и новые векторы в A догоняют B. Репликация для векторных структур обычно асинхронная: так снижается задержка в боевом регионе, а резерв догоняет с небольшим лагом. При плановом переключении или аварии трафик переводят в

регион B; генератор там работает с локальной копией индекса, не выезжая в соседний регион по сети.

Ниже в регионе B — слой CDN/Edge-кэшей. Он прячет часть нагрузок «на краю»: статики, прогретые подсказки, типовые ответы и даже результаты ретривала с коротким TTL. Это ощутимо уменьшает латентность для пользователей, которые географически ближе к резервной площадке, и помогает переживать всплески. Под всей конструкцией лежит единый контур безопасности: WAF фильтрует нежелательные шаблоны запросов, IAM управляет доступом сервисов к данным и индексам, KMS отвечает за ключи шифрования как «в полёте», так и «на диске».

В регионе A под основной «трассой» находится подсистема наблюдаемости. Она собирает метрики (латентность в разрезе этапов: шлюз → LLM → ретривер → БД), логи и трассировки запросов. Это важно не только для расследования инцидентов: по этим данным настраивают кэширование, меняют параметры индексов, балансируют шарды и аккуратно прогревают рабочие наборы. Если графики показывают рост задержки на ретривере, можно расширить вычислительные ресурсы индекса или пересобрать его с другими параметрами — и это видно почти в реальном времени.

Проблемы и вызовы

Главные сложности лежат на стыке инженерии, права и эксплуатации. Генеративная модель может выдавать впечатляющие результаты, но любой сбой в цепочке «данные → индексы → инференс» мгновенно превращается в ухудшение качества ответа, рост задержек и регуляторные риски. Ниже — ключевые зоны, с которыми неизбежно сталкивается команда.

Правовой статус данных и управление доступом. Разнородные корпуса тянут за собой лицензионные ограничения, обязанности по удалению (DSAR), требования к хранению персональных данных и географическим границам. Ошибка здесь проявляется не сразу: система работала месяцами, пока кто-то не запросил удаление или не выяснилась некорректная лицензия. Поэтому в хранилище должны жить не только сами документы, но и паспорт набора (datasetcard), отметки о происхождении, хэш-суммы и версия. Для персональных данных привычный «поисково-заменительный» подход малоэффективен: лучше использовать канонические формы хранения (токенизация с возможностью обратной развязки под контролем KMS) и атрибутивный контроль доступа на уровне строк/колонок, чтобы RAG не

вытягивал лишнее в контекст. Безопасность и враждебные воздействия. Генеративные стеки уязвимы на нескольких слоях. В источники могут попадать «отравленные» документы: малозаметные триггеры, которые меняют поведение модели при встрече с конкретной фразой. В контексте RAG часто встречается prompt-injection прямо в извлечённых фрагментах: текст «убедительно» просит игнорировать инструкции и сделать что-то иное. Есть и атакующие методы наподобие membershipinference (попытка понять, был ли конкретный пример в обучении) или modelinversion (восстановление приватных фрагментов). Практика показывает, что лучшим ответом остаётся многоступенчатый контур: подписанные источники и карантин новых потоков, дедупликация и эвристики токсичности на этапе приёма, «песочница» для ретривера (разрешённый подмножество правил), а также пост-фильтрация ответа. Для диагностики полезны канареечные документы и периодические «красные команды», которые целенаправленно пытаются сломать политику.

Качество корпусов и дрейф. Даже идеально построенный индекс устаревает. Тематические пропорции сдвигаются, свежие факты не попадают в выборку, а старые — продолжает находить ретривер. Симптомы понятны: растёт доля «почти релевантных» фрагментов, увеличивается объём безопорных рассуждений модели. Борются с этим метриками и регламентом: измеряют recall@K и nDCG на эталонных запросах, следят за совпадением даты документа с датой запроса, заставляют систему «отказываться» (abstain), если уверенность ниже порога, а затем планово переиндексируют слой за слоем: горячий кеш — ежедневно, основные сегменты — по расписанию, архив — реже.

Эксплуатация векторных индексов. Индексы дают скорость ценой памяти и сложной настройки. HNSW «ест» ОЗУ и требует аккуратного подбора параметров (M, efConstruction, efSearch): увеличишь — вырастет точность и латентность; уменьшишь — потеряешь документы. IVF/PQ экономит память и ускоряет кандидатов, но вносит квантовочную погрешность. В продакшене это решают ступенчато: грубый отбор на IVF/PQ, затем точный пересчёт расстояний (rerank) на небольшом наборе кандидатов. Отдельная боль — переиндексация. Полная перестройка на многомиллиардном корпусе занимает часы; поэтому делают rolling-переиндексацию, временно держат две версии индекса и переключают маршрутизацию по готовности. В распределённых развёртываниях ловят лаг репликации между регионами и «расхождение» шардов; помогает только телеметрия и регулярные сверки контрольных сумм.

Производительность и стоимость. Векторные витрины растут быстрее, чем «обычные» БД: один документ порождает десятки фрагментов, каждый — свой embedding. Память и диски набухают, бюджет на GPU для ANN-поиска упирается в потолок. Здесь нет серебряной пули, но работает набор приземлённых мер: агрессивная дедупликация «почти дублей», квантование до 8/4 бит там, где это приемлемо, хранение «холодных» сегментов в сжатом виде, жёсткие TTL на кэшах и отказ от бессрочных логов. Энергетическая часть тоже чувствительна: ночные окна для тяжёлых задач, планировщики с учётом углеродного следа, перенос вычислений в регионы с более «зелёной» генерацией.

Этика и контентная безопасность. Даже с чистыми данными модель может усиливать предвзятость, воспроизводить токсичную лексику или выдавать медицинские рекомендации без оснований. Здесь помогают не только фильтры, но и организационные меры: явные рубежи ответственности (что система не делает), журнал причин отказа, внутренние процедуры эскалации для спорных случаев. Для пользовательских сценариев с детьми, финансами или медициной разумно вводить двухступенчатые ответы: сначала фактологическая справка из RAG, затем — «человеческая» часть, где модель аккуратно формулирует выводы и предлагает источник.

Практика показывает, что успешные внедрения строятся на сочетании трёх классов хранилищ: транзакционного, документного и векторного. Такой состав позволяет обеспечить управляемость, масштаб и способность к семантическому поиску. Дальнейшее развитие пойдёт по пути автономизации, повышения безопасности и углубления интеграции с вычислительными средами.

Список литературы

1. Garcia-Molina, H.; Ullman, J.; Widom, J. Database Systems: The Complete Book. – 2nd ed. – Pearson, 2023. – 1248 p.

2. Stonebraker, M.; Hellerstein, J. M. Readings in Database Systems (5th ed.). – MIT Press, 2024. – 620 p.

3. Pinecone. Vector Database for Machine Learning. – Режимдоступа: https://www.pinecone.io/, свободный. – Датаобращения: 10.09.2025.

4. Weaviate. Open-SourceVectorDatabase. – Режим доступа: https://weaviate.io/, свободный. – Дата обращения: 09.09.2025.

5. Milvus. Open-SourceVectorDatabaseforAI. – Режим доступа: https://milvus.io/, свободный. – Дата обращения: 12.09.2025.

6. FAISS (FacebookAISimilaritySearch). – Режим доступа: https://github.com/facebookresearch/faiss, свободный. – Дата обращения: 15.09.2025.

7. TimescaleInc. TimescaleDBDocumentation. – Режим доступа: https://docs.timescale.com/, свободный. – Дата обращения: 15.09.2025.

8. InfluxData. InfluxDBDocumentation – Режим доступа: https://docs.influxdata.com/, свободный. – Дата обращения: 15.09.2025.

9. OpenAI. Retrieval-AugmentedGeneration: BestPractices. – Режим доступа: https://platform.openai.com/, свободный. – Дата обращения: 15.09.2025.

References

1. Garcia-Molina, H., Ullman, J., & Widom, J. Database Systems: The Complete Book (2nd ed.). Pearson, 2023. – 1248 p.

2. Stonebraker, M., & Hellerstein, J. M. Readings in Database Systems (5th ed.). MIT Press, 2024. – 620 p.

3. Pinecone. Vector Database for Machine Learning. Available at: https://www.pinecone.io/. Accessed: September 10, 2025.

4. Weaviate. Open-Source Vector Database. Available at: https://weaviate.io/. Accessed: September 9, 2025.

5. Milvus. Open-Source Vector Database for AI. Available at: https://milvus.io/. Accessed: September 12, 2025.

6. FAISS (Facebook AI Similarity Search). Available at: https://github.com/facebookresearch/faiss. Accessed: September 15, 2025.

7. Timescale Inc. TimescaleDB Documentation. Available at: https://docs.timescale.com/. Accessed: September 15, 2025.

8. InfluxData. InfluxDB Documentation. Available at: https://docs.influxdata.com/. Accessed: September 15, 2025.

9. OpenAI. Retrieval-Augmented Generation: Best Practices. Availableat: https://platform.openai.com/. Accessed: September 15, 2025.