RuStore

Александр Юдин

Сергей Кузнецов

AI в цифровом производстве

План

1 Цели

2 Инструменты

3 Безопасность

4 Итоги

Цели

Наши цели

1. Увеличить перформанс инженеров, путем увеличения количества выполняемых задач через AI.

Наши цели

1. Увеличить перформанс инженеров, путем увеличения количества выполняемых задач через AI.

1. Переложить большую часть рутинной работы: тесты, код-ревью, поиск и разбор багов, и т.д. на AI, не ухудшая качества результата.

Наши цели

1. Увеличить перформанс инженеров, путем увеличения количества выполняемых задач через AI.

1. Переложить большую часть рутинной работы: тесты, код-ревью, поиск и разбор багов, и т.д. на AI, не ухудшая качества результата.

1. Дать каждому инженеру: доступный, безопасный и эффективный в задачах **AI инструмент на каждый день**.

Необходимые инструменты

1. LLMProxy

Необходимые инструменты

1. LLMProxy
2. AI Плагин в IDE или AI IDE

Необходимые инструменты

1. LLMProxy
2. AI Плагин в IDE или AI IDE
3. Агент code-review

LLM Proxy

Цель LLM Proxy

1) Предоставить возможность безопасного использования облачных провайдеров с топовыми моделями для инженеров компании.

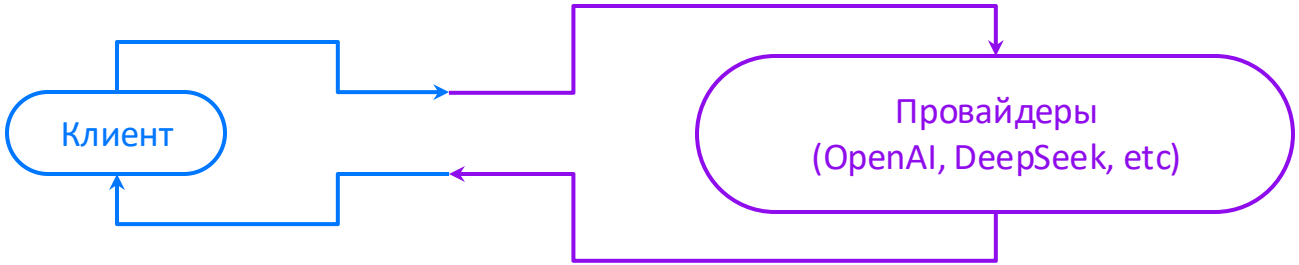
Цель LLM Proxy

1) Предоставить возможность безопасного использования облачных провайдеров с топовыми моделями для инженеров компании.
2) Вести контроль бюджетов использования LLM

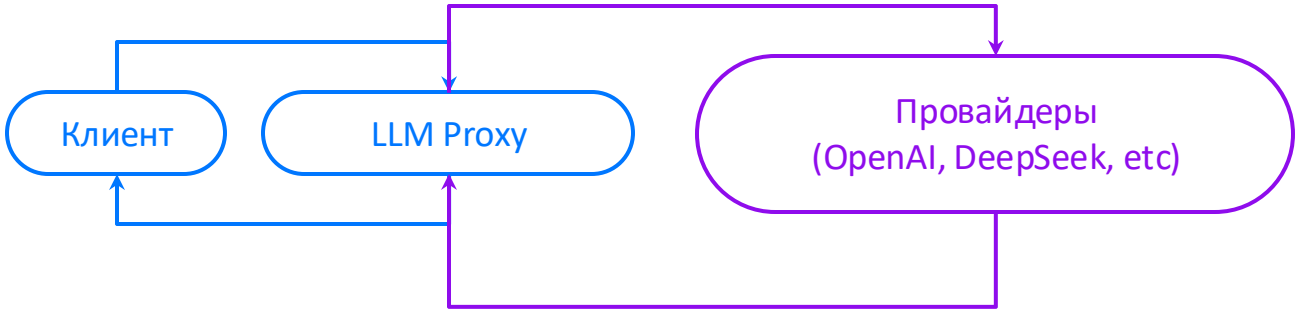
Цель LLM Proxy

1) Предоставить возможность безопасного использования облачных провайдеров с топовыми моделями для инженеров компании.
2) Вести контроль бюджетов использования LLM
3) Сохранение промптов для возможности анализа ИБ

Облачные провайдеры



Облачные провайдеры



Проблемы облачных провайдеров

1) Часть провайдеров недоступна без ВПН

Проблемы облачных провайдеров

1) Часть провайдеров недоступна без ВПН
2) Рейт лимиты аккаунтов

Проблемы облачных провайдеров

1) Часть провайдеров недоступна без ВПН
2) Рейт лимиты аккаунтов
3) Сведение бюджетов в рамках БЮ/Компании/Отделов

Решение доступности

1) Набор VDS роутов

Решение рейт лимитов

1) Заведение “сервисных” ключей в рамках БЮ

Решение рейт лимитов

1) Заведение “сервисных” ключей в рамках БЮ
2) Пользователи получают личный ключ

Решение рейт лимитов

1) Заведение “сервисных” ключей в рамках БЮ
2) Пользователи получают личный ключ
3) Пользователям доступны только модели их БЮ

Решение рейт лимитов

1) Заведение “сервисных” ключей в рамках БЮ
2) Пользователи получают личный ключ
3) Пользователям доступны только модели их БЮ
4) Распространение конфигураций с учетом нагрузки

Где деньги?

Где деньги?

1) Каждый БЮ имеет месячный лимит в $

Где деньги?

1) Каждый БЮ имеет месячный лимит в $
2) Каждый сотрудник имеет дневной лимит в $

Где деньги?

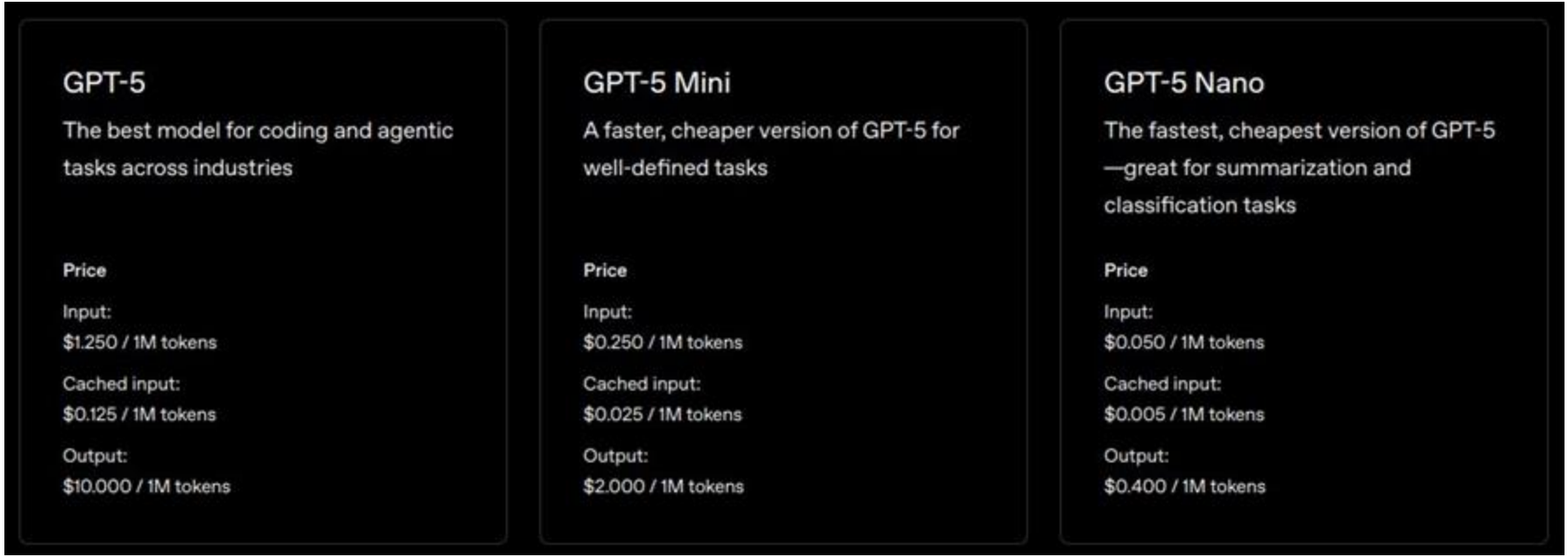
1) Каждый БЮ имеет месячный лимит в $
2) Каждый сотрудник имеет дневной лимит в $
3) Нужно вести правильную калькуляцию использования

Как считать?

Как считать?

```
"usage": {
    "prompt_tokens": 25,
    "completion_tokens": 3413,
    "total_tokens": 3438,
    "prompt_tokens_details": {
        "cached_tokens": 0,
    },
    "completion_tokens_details": {
        "reasoning_tokens": 2624,
    }
```

Как считать?



GPT-5

The best model for coding and agentic tasks across industries

Price

Input:
$1.250 / 1M tokens

Cached input:
$0.125 / 1M tokens

Output:
$10.000 / 1M tokens

GPT-5 Mini

A faster, cheaper version of GPT-5 for well-defined tasks

Price

Input:
$0.250 / 1M tokens

Cached input:
$0.025 / 1M tokens

Output:
$2.000 / 1M tokens

GPT-5 Nano

The fastest, cheapest version of GPT-5 —great for summarization and classification tasks

Price

Input:
$0.050 / 1M tokens

Cached input:
$0.005 / 1M tokens

Output:
$0.400 / 1M tokens

Как считать?

```
"usage": {
    "prompt_tokens": (25 - cached_tokens) * input price,
    "completion_tokens": 3413,
    "total_tokens": 3438,
    "prompt_tokens_details": {
        "cached_tokens": 0,
    },
    "completion_tokens_details": {
        "reasoning_tokens": 2624,
}
```

Как считать?

```
"usage": {
    "prompt_tokens": (25 - cached_tokens) * input price,
    "completion_tokens": 3413 * output price,
    "total_tokens": 3438,
    "prompt_tokens_details": {
        "cached_tokens": 0,
    },
    "completion_tokens_details": {
        "reasoning_tokens": 2624,
}
```

Как считать?

```
"usage": {
    "prompt_tokens": (25 - cached_tokens) * input price,
    "completion_tokens": 3413 * output price,
    "total_tokens": 3438,
    "prompt_tokens_details": {
        "cached_tokens": 0 * cached input price,
    },
    "completion_tokens_details": {
        "reasoning_tokens": 2624,
}
```

Итоги

1) Распределение сервисных аккаунтов на всю компанию

Итоги

1) Распределение сервисных аккаунтов на всю компанию
2) Контроль расходов

Итоги

1) Распределение сервисных аккаунтов на всю компанию
2) Контроль расходов
3) Логирование данных

Итоги

1) Распределение сервисных аккаунтов на всю компанию
2) Контроль расходов
3) Логирование данных
4) Единая точка входа в LLM

IDE + Инструменты

Плагины:

1. Cursor

Плагины:

1. Cursor
2. Cline (Плагин для VSCode)

Плагины:

1. Cursor
2. Cline (Плагин для VSCode)
3. RooCode (Плагин для VSCode)

Плагины:

1. Cursor
2. Cline (Плагин для VSCode)
3. RooCode (Плагин для VSCode)
4. Continue (Cross-Ide плагин VSCode + IDEA)

Плагины:

1. Cursor
2. Cline (Плагин для VSCode)
3. RooCode (Плагин для VSCode)
4. Continue (Cross-Ide плагин VSCode + IDEA)

Cursor:

+ Очень хорошо настроен агент

Cursor:

+ Очень хорошо настроен агент
+ Качественный автокомплит

Cursor:

+ Очень хорошо настроен агент
+ Качественный автокомплит
- Это форк VSCode == плохая поддержка Kotlin/Java

Cursor:

+ Очень хорошо настроен агент
+ Качественный автокомплит
- Это форк VSCode == плохая поддержка Kotlin/Java
- Нужно использовать 2 IDE (Android Studio + Cursor)

Cursor:

+ Очень хорошо настроен агент
+ Качественный автокомплит
- Это форк VSCode == плохая поддержка Kotlin/Java
- Нужно использовать 2 IDE (Android Studio + Cursor)
- Весь проект попадает под индексацию и отправляется в облако

Cline:

+ Хороший итеративный агент

Cline:

+ Хороший итеративный агент
- Автодополнение не работает

Cline:

+ Хороший итеративный агент
- Автодополнение не работает
- Ограничена кастомизация настроек модели

Cline:

+ Хороший итеративный агент
- Автодополнение не работает
- Ограничена кастомизация настроек модели
- Очень высокий расход токенов

Cline:

+ Хороший итеративный агент
- Автодополнение не работает
- Ограничена кастомизация настроек модели
- Очень высокий расход токенов
- Нет возможности использовать разные модели для разных ситуаций

Cline:

+ Хороший итеративный агент
- Автодополнение не работает
- Ограничена кастомизация настроек модели
- Очень высокий расход токенов
- Нет возможности использовать разные модели для разных ситуаций
- Это VSCode

RooCode (fork Cline):

+ Хороший итеративный агент

RooCode (fork Cline):

+ Хороший итеративный агент
+ Кастомизация настроек модели расширена относительно Cline

RooCode (fork Cline):

+ Хороший итеративный агент
+ Кастомизация настроек модели расширена относительно Cline
+ Хорошая кастомизация промптов

RooCode (fork Cline):

+ Хороший итеративный агент
+ Кастомизация настроек модели расширена относительно Cline
+ Хорошая кастомизация промптов

RooCode (fork Cline):

+ Хороший итеративный агент
+ Кастомизация настроек модели расширена относительно Cline
+ Хорошая кастомизация промптов
- Автодополнение не работает

RooCode (fork Cline):

+ Хороший итеративный агент
+ Кастомизация настроек модели расширена относительно Cline
+ Хорошая кастомизация промптов
- Автодополнение не работает
- Это VSCode

Continue

+ Наличие агента

Continue

+ Наличие агента
+ Очень высокая кастомизация

Continue

+ Наличие агента
+ Очень высокая кастомизация
+ Можно настроить какая модель за что отвечает

Continue

+ Наличие агента
+ Очень высокая кастомизация
+ Можно настроить какая модель за что отвечает
+ Автодополнение кода в редакторе

Continue

+ Наличие агента
+ Очень высокая кастомизация
+ Можно настроить какая модель за что отвечает
+ Автодополнение кода в редакторе
+ Плагин работает как в VSCode, так и в IDEA

Continue

+ Наличие агента
+ Очень высокая кастомизация
+ Можно настроить какая модель за что отвечает
+ Автодополнение кода в редакторе
+ Плагин работает как в VSCode, так и в IDEA
- В IDEA работает через JCEF

Continue

+ Наличие агента
+ Очень высокая кастомизация
+ Можно настроить какая модель за что отвечает
+ Автодополнение кода в редакторе
+ Плагин работает как в VSCode, так и в IDEA
- В IDEA работает через JCEF
- Сложная реализация, высокий порог входа для котрибьюта (Плагин реализован на трех основных языках: Kotlin, TypeScript, Python)

Итоги обзора плагинов:

1) Cursor лидер в автоматизации разработки, но “забирает” Ваш проект к себе. Вторая IDE
2) Cline и RooCode хороши для OpenSource решений, но немного отстают от Cursor. Вторая IDE

Итоги обзора плагинов:

3) Continue работает в IDEA мире, но уступает в качестве агентских возможностей. Высокая кастомизируемость может показаться избыточной, но понравится linux пользователям)
4) Continue и RooCode позволяют распространять настройки централизованно

Code-review agent

Что такое code-review agent

Сервис для проведения code-review с помощью LLM

Проблемы

1) Ручной code-review - долго и дорого, зависит от человеческого фактора

Проблемы

1) Ручной code-review - долго и дорого, зависит от человеческого фактора
2) Мелкие ошибки и опечатки могут проскальзывать

Проблемы

1) Ручной code-review - долго и дорого, зависит от человеческого фактора
2) Мелкие ошибки и опечатки могут проскальзывать
3) Нужно ускорить метрики TTM без снижения качества

Решение

1) LLM-агент, который анализирует изменения в Merge Request

Решение

1) LLM-агент, который анализирует изменения в Merge Request
2) Комментирует код прямо в GitLab

Решение

1) LLM-агент, который анализирует изменения в Merge Request
2) Комментирует код прямо в GitLab
3) Легкая интеграция в CI/CD

Ключевые возможности

1) Поддержка нескольких режимов работы

Ключевые возможности

1) Поддержка нескольких режимов работы
    a) llm-single

Ключевые возможности

1) Поддержка нескольких режимов работы
    a) llm-single
    b) ReAct agent (думаю -> действую -> наблюдаю)

Ключевые возможности

1) Поддержка нескольких режимов работы
    a) llm-single
    b) ReAct agent (думаю -> действую -> наблюдаю)
    c) ToolCalling (нативный вызов инструментов)

Ключевые возможности

1) Поддержка нескольких режимов работы
    a) llm-single
    b) ReAct agent (думаю -> действую -> наблюдаю)
    c) ToolCalling (нативный вызов инструментов)
    d) Planning agent

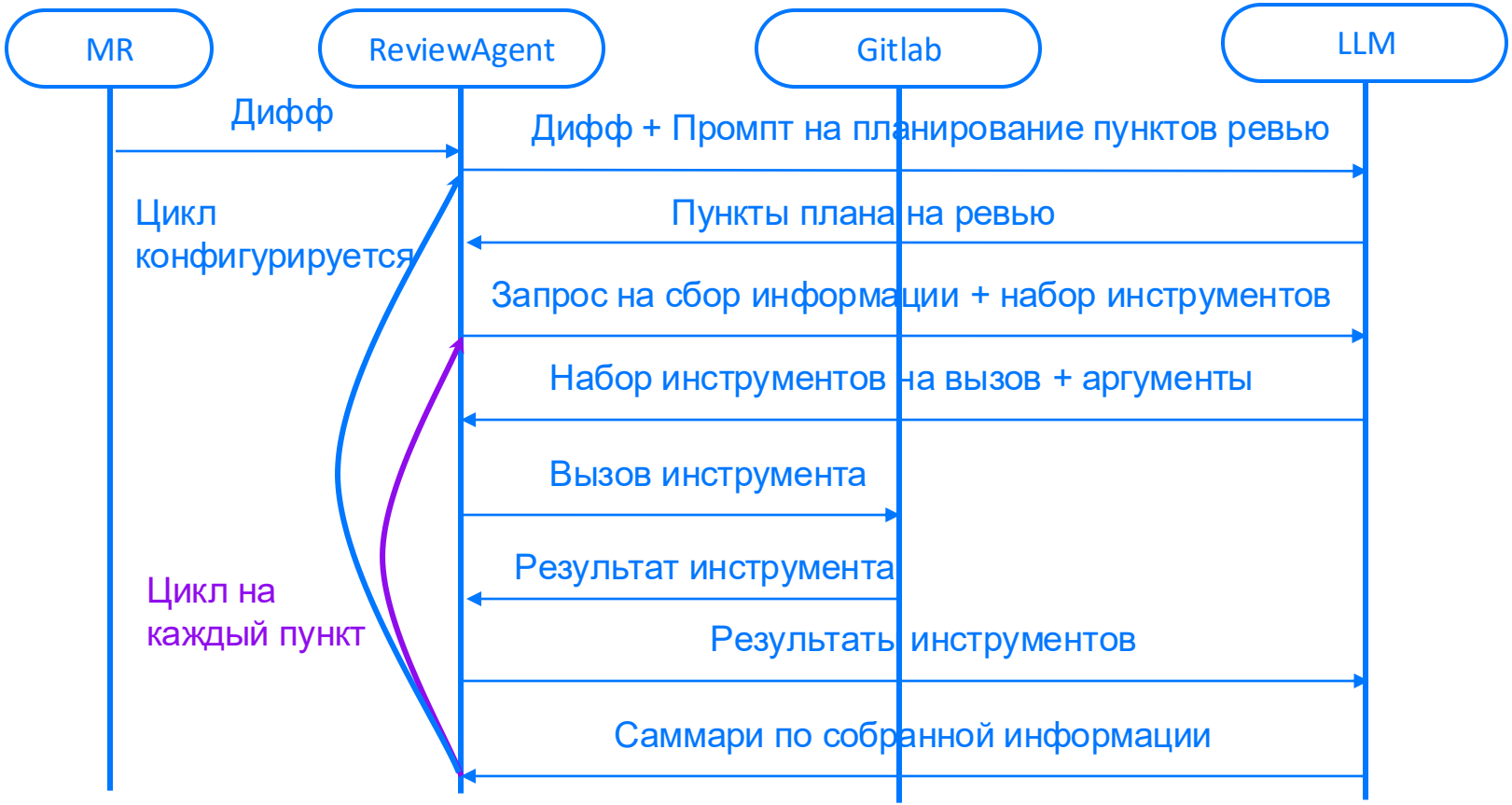
Ключевые возможности

1) Поддержка нескольких режимов работы
    a) llm-single
    b) ReAct agent (думаю -> действую -> наблюдаю)
    c) ToolCalling (нативный вызов инструментов)
    d) Planning agent
2) Гибкая настройка моделей, поддерживает любую модель в OpenAI формате

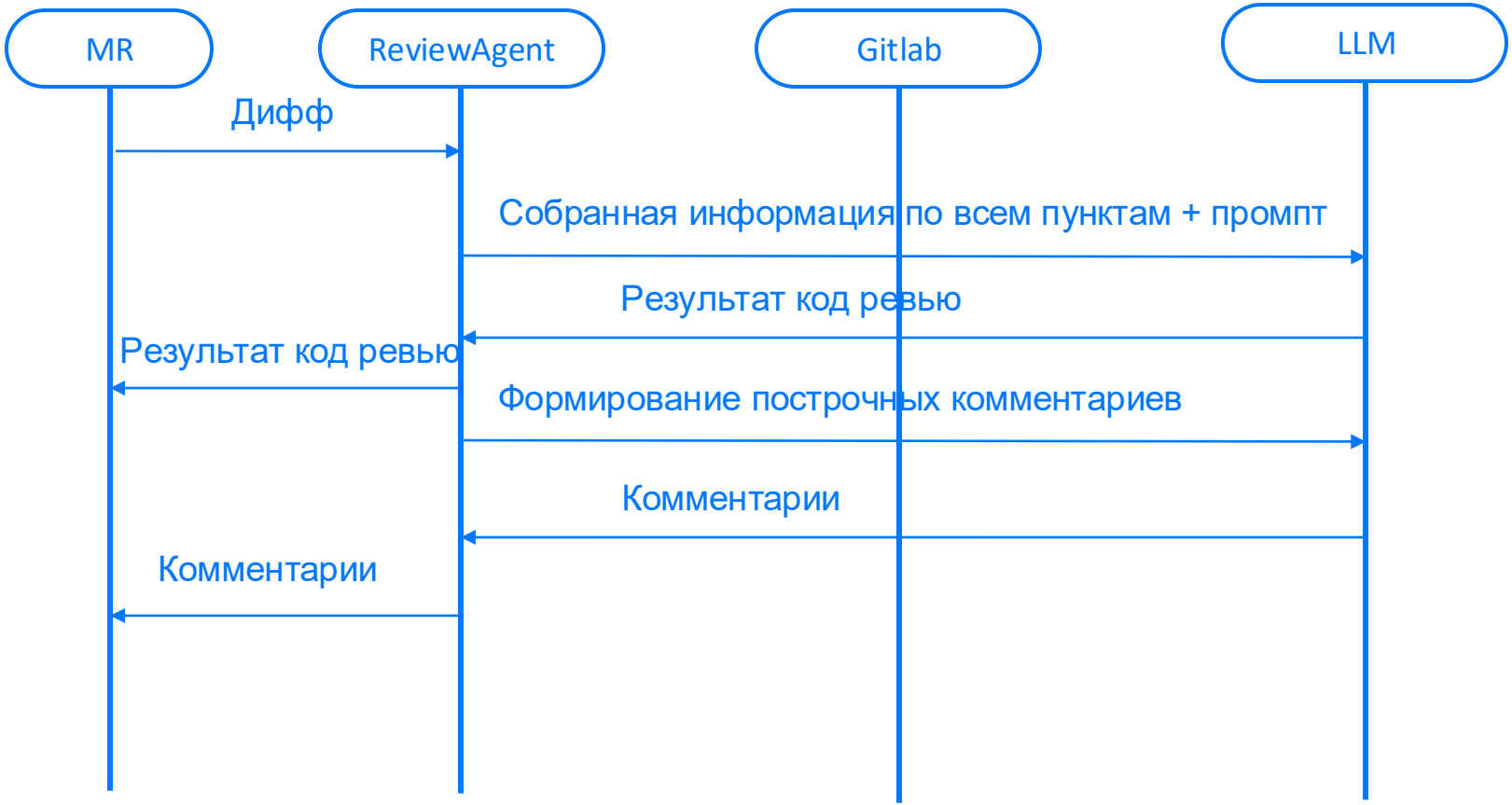
Ключевые возможности

1) Поддержка нескольких режимов работы
    a) llm-single
    b) ReAct agent (думаю -> действую -> наблюдаю)
    c) ToolCalling (нативный вызов инструментов)
    d) Planning agent
2) Гибкая настройка моделей, поддерживает любую модель в OpenAI формате
3) Комментарии в виде общего саммари + замечания по строкам.

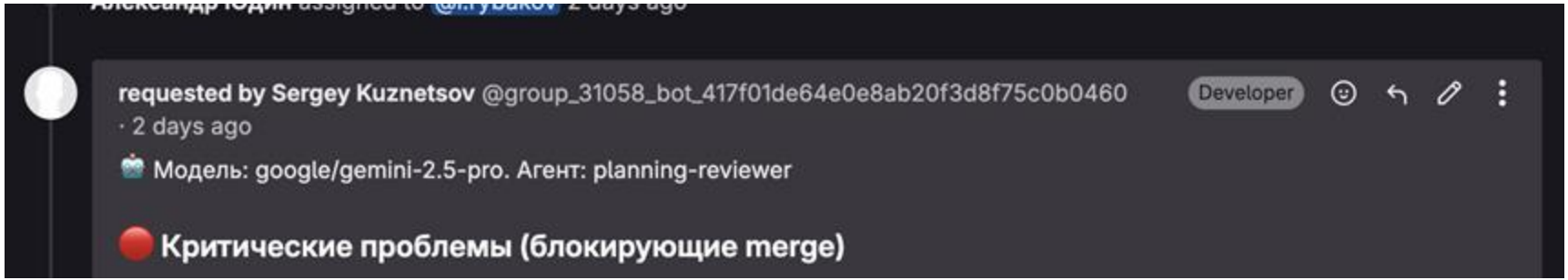
Planning agent



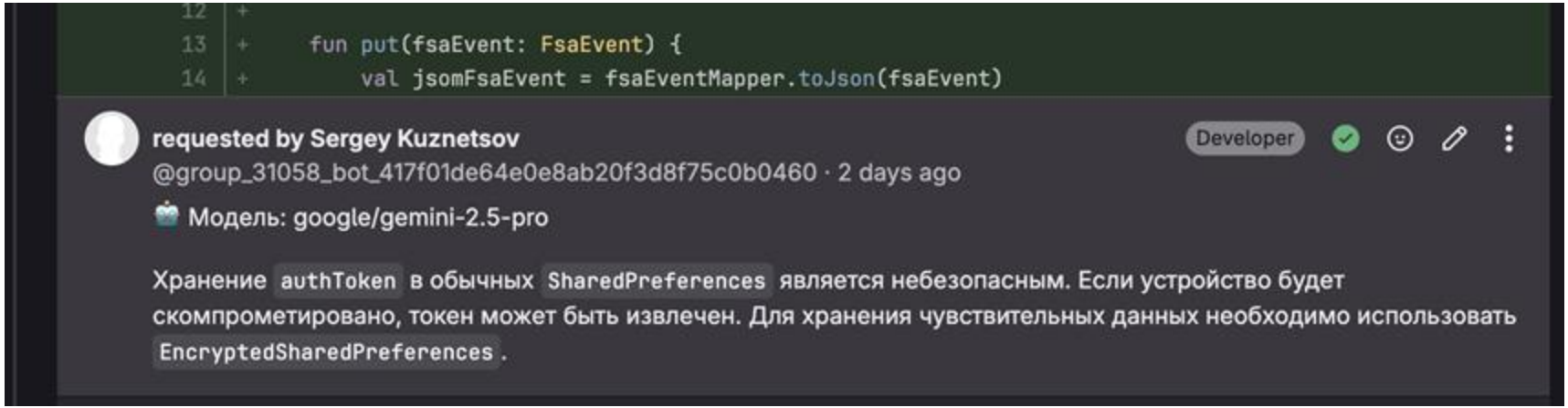
Planning agent



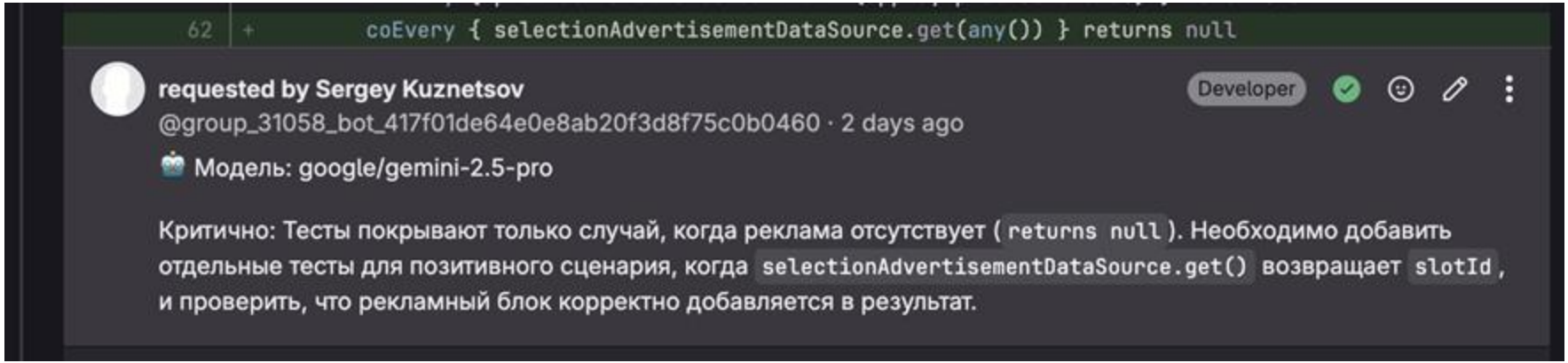
Planning agent



Planning agent



Planning agent



Итоги внедрения Review-agent

1) Сокращение времени ревью
2) Покрытие “рутиной”: стилистика, мелкие ошибки, пропущенные проверки
3) Люди концентрируются на бизнес логике и архитектуре
4) Помогает с унификацией стандартов кода

Безопасность RuStore LLM Proxy

Безопасность RuStore LLM Proxy

Для обеспечения безопасного взаимодействия с внешними провайдерами LLM был развернут сервис LLM Guard

LLM Guard — инструмент для обеспечения безопасности LLM-приложений путем анализа входных промптов и ответов модели

В контексте RuStore LLM Proxy сервис используется для маскирования конфиденциальных данных (секреты, ПнД, etc) в пользовательских промптах, а также для ограничение количества токенов в одном промпте - до 128 тысяч токенов

Проблемы безопасности при работе с LLM

Взаимодействие с LLM несет значительные риски, требующие немедленного внимания:

Утечка чувствительных данных

- Персональные и корпоративные данные в промптах.
- Передача API ключей и токенов в открытом виде.

Требования обработки в реальном времени

- Необходимость быстрой санитизации для стриминговых ответов.
- Поддержание производительности и минимальных задержек.

Последствия включают нарушение требований безопасности и потенциальные утечки конфиденциальной информации.

Что решает система обфускации и деобфускации кода

Наша система обфускации нацелена на решение ключевых проблем безопасности LLM:

Автоматическая санитизация

Промпты автоматически очищаются перед отправкой к LLM, предотвращая утечки данных.

Деобфускация ответов

Восстановление исходного контекста ответов для обеспечения прозрачности для пользователя.

Поддержка стриминга

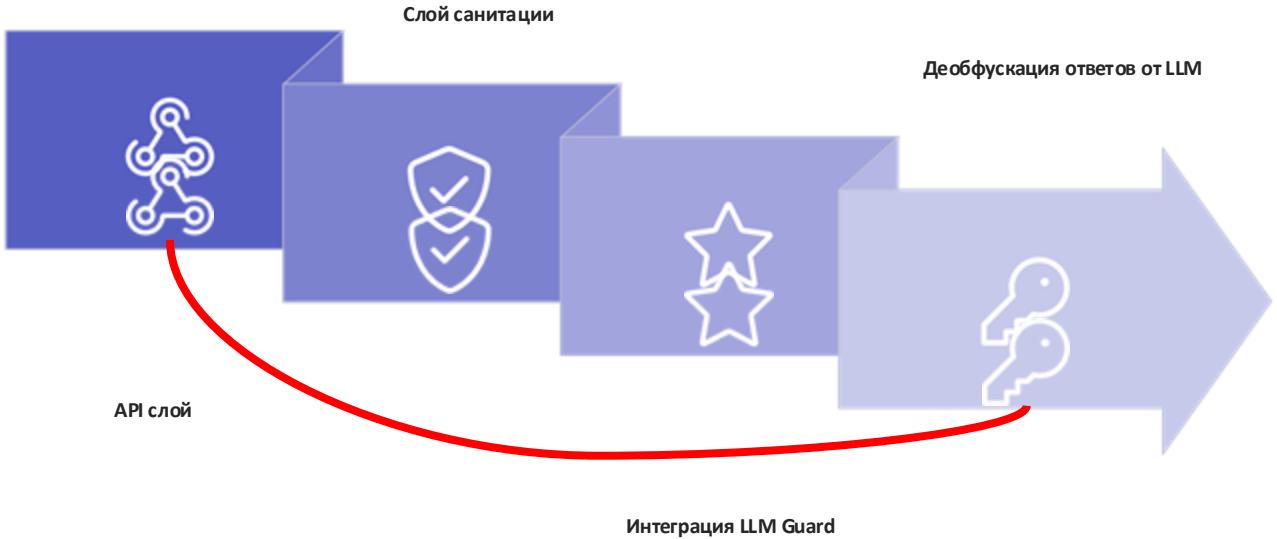
Обработка данных в реальном времени для стриминговых потоков, минимизируя задержки.

Гибкая конфигурация

Настраиваемые правила санитизации для адаптации к различным сценариям использования.

Общая архитектура системы

Представляем общую архитектуру системы обфускации и деобфускации, интегрированную с LLM Guard.



Эта архитектура обеспечивает бесшовную интеграцию и надежную защиту данных на всех этапах взаимодействия с LLM.

Ключевые компоненты системы

Система состоит из нескольких взаимосвязанных компонентов, обеспечивающих полный цикл обработки данных от обфускации промтов до десанитизации ответов от llm:

API Layer

Предоставляет внешние точки входа для запросов (`/v1/chat/completions`, `/v1/completions` `v1/responses`) и управляет обработкой стриминговых и не-стриминговых потоков.

Слой санитизации промтов

Включает `PromptSanitizationService` для координации процесса очистки и `GuardService` для интеграции с LLM Guard.

LLM Guard сервис

Внешний сервис, содержащий набор специализированных сканеров (Regex, Secrets, Token Limit) для анализа и санитизации промптов.

Слой деобфускации ответов от LLM

Состоит из `DeobfuscationProcessor`, отвечающего за стриминговую деобфускацию, и Vault системы для безопасного хранения соответствий между маскированными и исходными данными.

Общая диаграмма процесса обфускации и деобфускации

Система обеспечивает сквозную защиту данных, от момента получения запроса до деобфускации стримингового ответа LLM. Этот комплексный подход гарантирует конфиденциальность информации на каждом этапе взаимодействия.

Инициация запроса

Клиент отправляет стриминговый запрос в API.

Санитизация промптов

API вызывает службу PromptSanitizationService (PSS) для обработки входных данных.

Обработка LLM Guard

PSS отправляет данные в LLM Guard для анализа и обфускации чувствительной информации.

Получение Vault и ответа

LLM Guard возвращает обфусцированные промпты и "Vault" – хранилище оригинальных данных.

Запрос к LLM-провайдеру

API, используя "Vault", передает запрос LLM-провайдеру.

Стриминг ответа и деобфускация

Провайдер стримит ответ, который проходит через DeobfuscationProcessor для восстановления исходных данных с использованием "Vault".

Финальный ответ клиенту

Деобфусцированные фрагменты ответа передаются клиенту в реальном времени, обеспечивая непрерывность и безопасность.

Эта последовательность шагов демонстрирует, как система интегрирует процессы санитизации и деобфускации для обеспечения безопасного и эффективного взаимодействия с LLM в стриминговом режиме.

Слой санитизации промтов

`PromptSanitizationService` играет центральную роль в управлении процессом санитизации:

```
async def sanitize_prompts(self, data: dict, request_id: str, request: Request) -> dict:
    """ Основной метод для внешней санитизации через LLM Guard """
    sanitized_data = copy.deepcopy(data)
    prompts = await self._extract_prompts(data)

    # Проверка лимита токенов
    await self.guard_service.check_token_limit(data=data, request_id=request_id)

    # Подготовка промптов для санитизации
    prompts_for_sanitization = self._prepare_prompts(prompts)
    prompts_dict = {f"prompt_{i}": prompt for i, prompt in enumerate(prompts_for_sanitization)}

    # Отправка в LLM Guard
    batch_result = await self.guard_service.verify_completion_batch(
        data=data, prompts=prompts_dict, request_id=request_id
    )

    # Сохранение Vault в состоянии запроса
    request.state.vault = batch_result["vault"]
    return self._update_request_data(sanitized_data, batch_result)
```

Этот метод координирует извлечение промптов, проверку лимитов токенов, отправку в LLM Guard и сохранение `Vault` для последующей деобфускации.

Слой санитизации промтов

```python
async def verify_completion_batch(self, data: dict, prompts: Dict[str, str], request_id: str) -> Dict[str, Any]:
    headers = {
        "Authorization": f"Bearer {self.llm_guard_api_key}",
        "Content-Type": "application/json",
        "X-Vault-Placeholder-Template": REDACTED_MASK  # REDACTED{n}
    }

    guard_request_data = {
        "prompts": list(prompts.values()),
        "scanners_select": ["Regex", "Secrets"]  # Выбор сканеров
    }

    response = await self.http_client.post(
        f"{self.llm_guard_url}/analyze/prompt",
        headers=headers,
        json=guard_request_data,
        timeout=30.0
    )

    if response.status_code != 200:
        error_message = await response.json()
        raise HTTPException(response.status_code, f"LLM Guard error: {error_message}")

    response_data = response.json()
    sanitized_prompts_list = response_data["sanitized_prompt"]

    # Восстанавливаем структуру словаря промптов
    sanitized_prompts_dict = {}
    prompt_keys = list(prompts.keys())

    for i, sanitized_value in enumerate(sanitized_prompts_list):
        if i < len(prompt_keys):
            sanitized_prompts_dict[prompt_keys[i]] = sanitized_value

    return {
        "sanitized_prompts": sanitized_prompts_dict,
        "scan_results": response_data["scanners"],
        "vault": response_data["vault"]
    }
```

Этот код демонстрирует, как промпты отправляются в LLM Guard для анализа и санитизации, а также как обрабатываются полученные результаты, включая сохранение `Vault`.

Метод `verify_completion_batch()` в `GuardService` обеспечивает бесшовное взаимодействие с внешним сервисом LLM Guard:

LLM Guard Сервис

LLM Guard предоставляет набор мощных сканеров для выявления и нейтрализации угроз безопасности:

Regex Scanner

Обнаруживает чувствительные данные по настраиваемым регулярным выражениям, включая JWT токены и API ключи.

Secrets Scanner

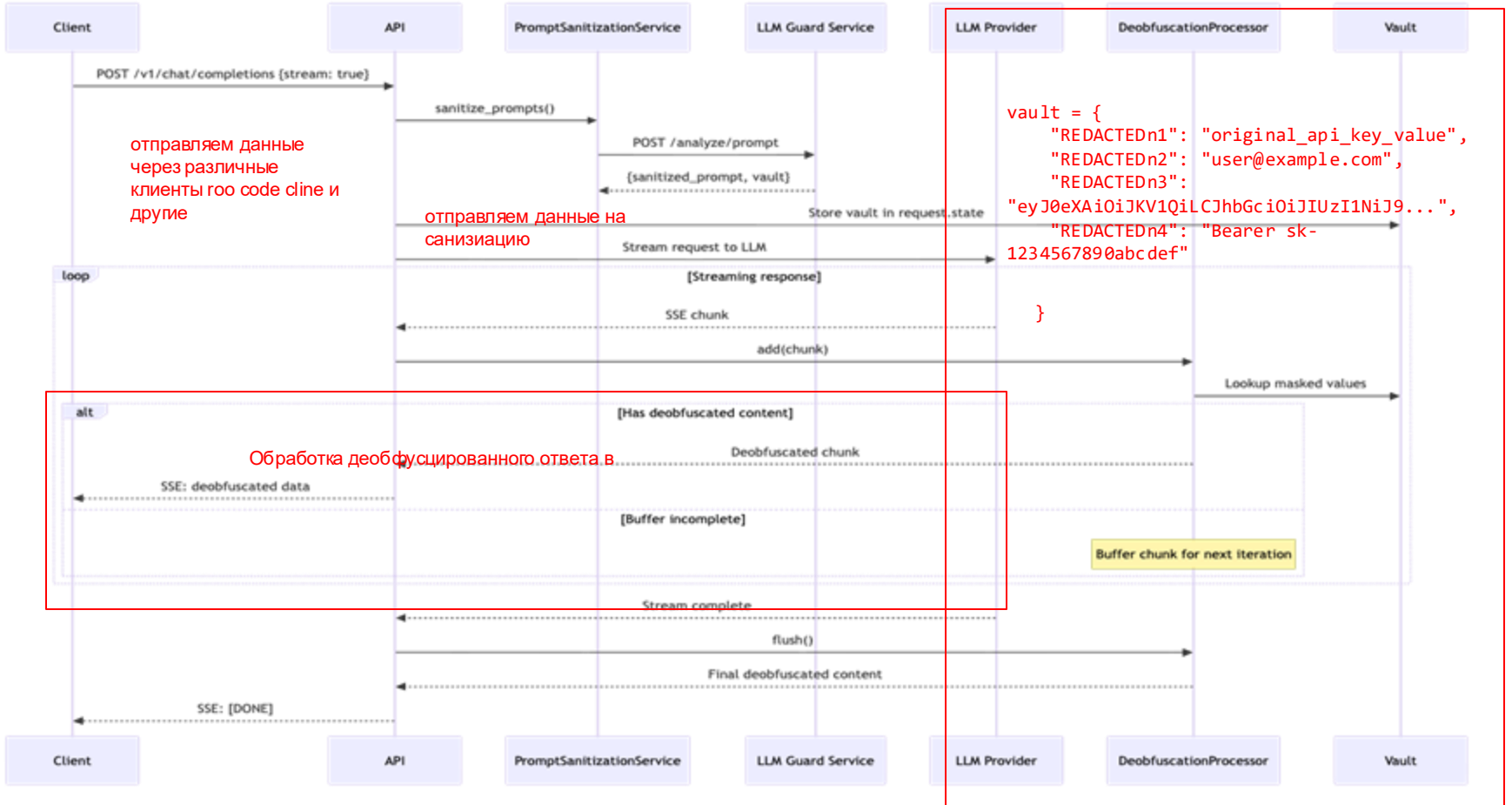
Идентифицирует различные типы секретов, такие как ключи AWS и учетные данные баз данных, благодаря интеграции с базами известных паттернов.

Token Limit Scanner

Предотвращает превышение лимитов токенов LLM, для предотвращения чрезмерной загрузки кода в облако

Конфигурация сканеров происходит через параметр `scanners_select` в запросе к LLM Guard, обеспечивая гибкость и контроль.

Слой деобфускации ответов от LLM



Слой деобфускации ответов от LLM

```
class DeobfuscationProcessor:
    def __init__(self, vault: Dict[str, str]):
        self.vault = vault
        self.max_key_len = max((len(k) for k in
vault), default=0)
        self.buffer: str = ""
        self.output_str: str = ""

    def add(self, chunk: str) -> None:
            self.buffer += chunk
            while True:
                match = PATTERN.search(self.buffer)  #
    Поиск REDACTEDn\d+
                if match:
                    key = match.group(0)
                    replacement = self.vault.get(key,
    key)
                    self.output_str +=
    self.buffer[:match.start()] + replacement
                    self.buffer =
    self.buffer[match.end():]
                    continue
                break

        def flush(self) -> str:
            # Финальная обработка остатков буфера
            raw = self.output_str + self.buffer
            return self._replace_all_patterns(raw)
```

Для эффективной обработки стриминговых ответов LLM в реальном времени, система использует компонент `DeobfuscationProcessor`. Он динамически восстанавливает исходные конфиденциальные данные из обфусцированных ответов, используя Vault.

Метод `add()` постепенно обрабатывает поступающие части ответа, ищет маскированные значения (по шаблону `REDACTEDn{n}`) и заменяет их на оригинальные данные из `Vault`. Метод `flush()` обеспечивает окончательную обработку буфера после завершения стриминга, гарантируя полную деобфускацию всего содержимого.

Практические примеры санизизации и десанитизации промтов и ответов от LLM

Пример 1: Санитизация API ключа

Исходный промпт:
"Используй этот API ключ: sk-1234567890abcdef для подключения к OpenAI"

После санитизации:
"Используй этот API ключ: REDACTEDn1 для подключения к OpenAI"

Vault:
{"REDACTEDn1": "sk-1234567890abcdef"}

Пример 2: Обработка JWT токена

Исходный промпт:
"Авторизуйся с токеном: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."

После санитизации:
"Авторизуйся с токеном: REDACTEDn2"

Vault:
{"REDACTEDn2": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."}

Пример 3: Стриминговая деобфускация

LLM ответ (chunks):
"Для подключения используйте" → "REDACTEDn1" → "в заголовке Authorization"

Деобфускированный результат:
"Для подключения используйте sk-1234567890abcdef в заголовке Authorization"

Доп требования по безопасности RuStore LLM Proxy

Общие требования для пользователей:

- Наличие LLM Guard не исключает требования на вынос секретов из исходного кода в ENV-переменные

Доп требования по безопасности RuStore LLM Proxy

Общие требования для пользователей:

- Наличие LLM Guard не исключает требования на вынос секретов из исходного кода в ENV-переменные
- Используйте только доверенные промпты. Некоторые массивные шаблоны промптов могут содержать бэкдоры, не всегда заметные невооруженным глазом

Доп требования по безопасности RuStore LLM Proxy

Общие требования для пользователей:

- Наличие LLM Guard не исключает требования на вынос секретов из исходного кода в ENV-переменные
- Используйте только доверенные промпты. Некоторые массивные шаблоны промптов могут содержать бэкдоры, не всегда заметные невооруженным глазом
- Ограничивайте отправляемые во внешние LLM данные только необходимым контекстом

Асинхронный мониторинг промтов пользователей

Независимо от LLM Guard запущен асинхронный мониторинг журнала пользовательских промптов с применением внутренней LLM в компании

Асинхронный мониторинг промтов пользователей

Независимо от LLM Guard запущен асинхронный мониторинг журнала пользовательских промптов, который, применяя VK LLM:

- Использует контекстный анализ для выявления злоупотреблений в выгрузке кодовой базы

Асинхронный мониторинг промтов пользователей

Независимо от LLM Guard запущен асинхронный мониторинг журнала пользовательских промптов, который, применяя VK LLM:

- Использует контекстный анализ для выявления злоупотреблений в выгрузке кодовой базы
- Детектирует утекшие конфиденциальные данные

Асинхронный мониторинг промтов пользователей

Независимо от LLM Guard запущен асинхронный мониторинг журнала пользовательских промптов, который, применяя VK LLM:

- Использует контекстный анализ для выявления злоупотреблений в выгрузке кодовой базы
- Детектирует утекшие конфиденциальные данные
- Оповещает пользователей о нарушениях и блокирует доступ к прокси в случае их неоднократного повторения

Итоги внедрения LLM Guard Service

Внедрение LLM Guard Service принесло значительные преимущества, укрепив безопасность и эффективность взаимодействия с моделями LLM.

Усиленная безопасность

Надежная защита конфиденциальных данных в промптах и ответах LLM от несанкционированного доступа.

Обработка в реальном времени

Бесшовная деобфускация стриминговых ответов без задержек, сохраняя высокую производительность.

Сохранность данных

Точное и безопасное восстановление исходных конфиденциальных данных в ответах LLM благодаря Vault.

Гибкая настройка

Возможность адаптации и расширения сканеров безопасности под уникальные требования проекта.

Это обеспечивает не только соответствие стандартам безопасности, но и повышает доверие к системе, позволяя использовать LLM с большей уверенностью.

Планы развития Rustore LLM Proxy + Guard Service

- Обфускация картинок перед передачей во внешние провайдеры LLM
- Реализация кастомных сканеров по обфускации кода проекта (SQL , python, kotlin, java и др) с сохранением структуры кода и контекста для LLM
- Реализация персисистетного хранилища замаскированных ключей в рамках сессии с провайдерам LLM

Истории успеха

Создание LLMProxy

Создали сервис не имея практического опыта работы с Python и его серверными компонентами

Сокращение времени разработки

1. Сокращает время разработки Unit тестов
2. Хорошо помогает выполнять типовые задачи
3. Отлично знает задокументированное поведение API (Например API Android OS или стабильные версии библиотек)

Фидбэк тестовой группы 1

- Конвертил картинки из SVG из Figma в Compose ImageVector

- закинул картинку в DeepSeek, получил ImageVector

- потратил 1 минут, ИИ справился с первого раза

- обычный конвертер не справился сгенерил кривую картинку, руками починить картинку не удалось

Фидбэк тестовой группы 2

Составил доку для "Стандартов по разработке SDK в RuStore".

- с ИИ сделал за 30 мин на основе анализа классов сдк

- без ИИ сделал бы за 3 часа

Фидбэк тестовой группы 3

Через плагин в студии можно скормить AI диф изменений (коммиты с MP) и попросить сделать ревью изменений. В результате получаем отчет по изменениям и места на что стоит обратить внимание.

Целиком ревьювера не заменяет и скорости не особо прибавляет. Из-за того, что пока не доверяю моделям на 100%. Но качество ревью улучшается, с учетом даже перехода на 1-го ревьювера.

Фидбэк тестовой группы 4

- накидать мясо для юнит тестов

попросил написать юниты на класс, результат не проходил, но чутка поправил и все прошло

заняло 15 минут

без ИИ писал бы 1-2 часа

Фидбэк тестовой группы 5

Закинул модуль Navigation и попросить объяснить что тут происходит

Получил внятный ответ, что это рукописная навигация как альтернатива Jetpack-овской и несколько рекомендаций

Заняло 3 минуты

Без этого заняло бы целый день

Фидбэк тестовой группы 6

- Нарисовал круговую диаграмму для фичи по статистике игр

- Сгенерировал через DeepSeek

- Скопировал в код и через 15 минут собрал драфт фичи

- Без ИИ заняло бы 2+ часов

Фидбэк тестовой группы 7

Написать сложный запрос в ClickHouse

Попросил написать сложный запрос с парсингом строк и нетривильной логикой выборки

Заняло 5 минут

Без ИИ заняло бы 2+ часов

Фидбэк тестовой группы 8

- Задача: в рамках разбора инца нужна была информация по структуре split APK и нарезке ресурсов.

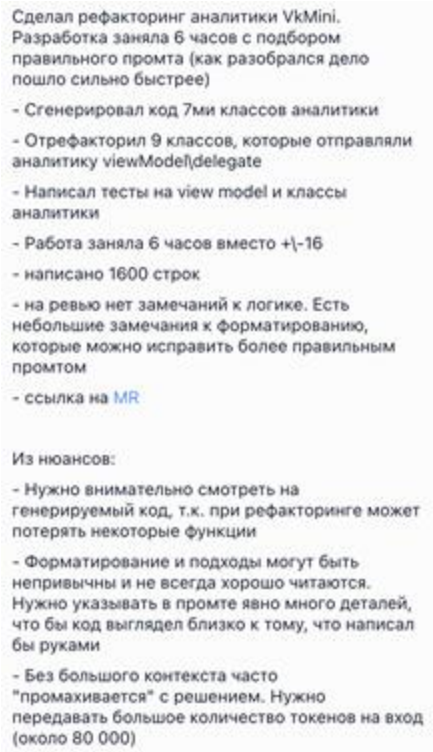
- ИИ дал совокупную информацию по механизму и сделал разбор под мой частный случай.

- С ИИ: вышло по времени 5 мин.

- БЕЗ ИИ: вышло бы 30 мин.

Фидбэк тестовой группы 9

Сделал рефакторинг аналитики VkMini с помощью LLM за 6 часов вместо 16, сгенерировал и отрефакторил 16 классов, написал 1600 строк кода. На ревью замечаний к логике не было. Важно внимательно проверять код и давать модели максимум контекста



Спасибо!