

МСР - единый "USB-C" протокол для ИИ приложений

Русинова Залина

Исследователь UDV Group,
Старший преподаватель УрФУ



Дата-завтрак сообщества ML ЕКВ,
Екатеринбург, 21.06.2025 г.

Содержание

Как ИИ общается с внешними системами?

Что такое MCP?

Протокол связи

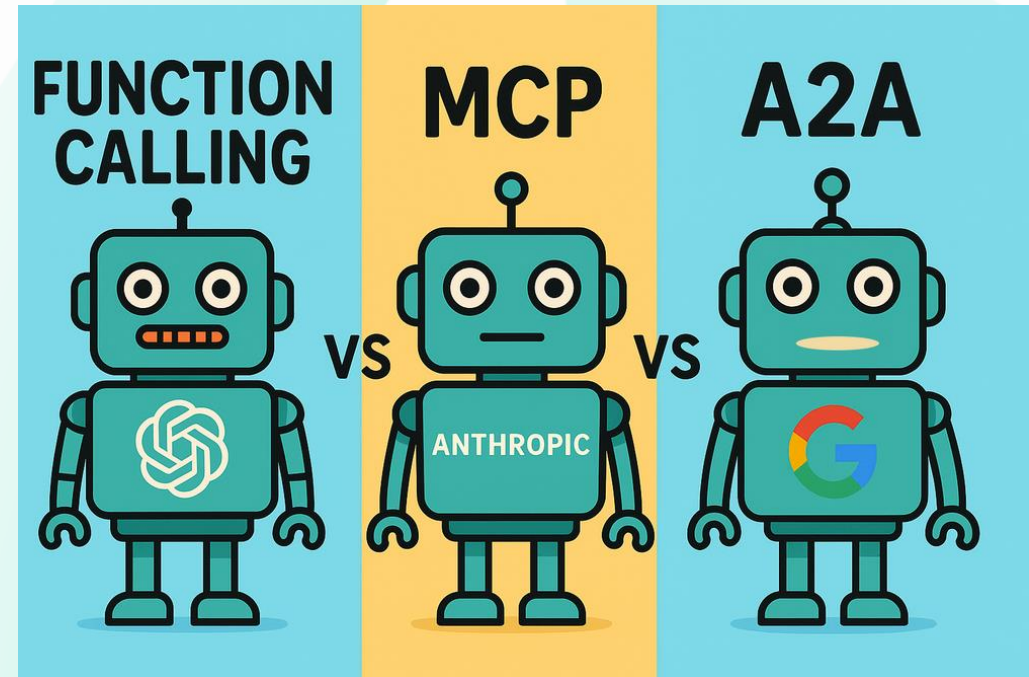
MCP SDK

DEMO

Подходы к построению архитектуры LLM-агентов

Как только вы начинаете создавать серьезные системы агентов с искусственным интеллектом, у вас возникает одна большая головная боль: нет четкого, универсального способа работы агентов с инструментами — или друг с другом.

- ✓ **Function Calling**
- ✓ **MCP (Model Context Protocol)**
- ✓ **A2A (Agent-to-Agent Protocol)**

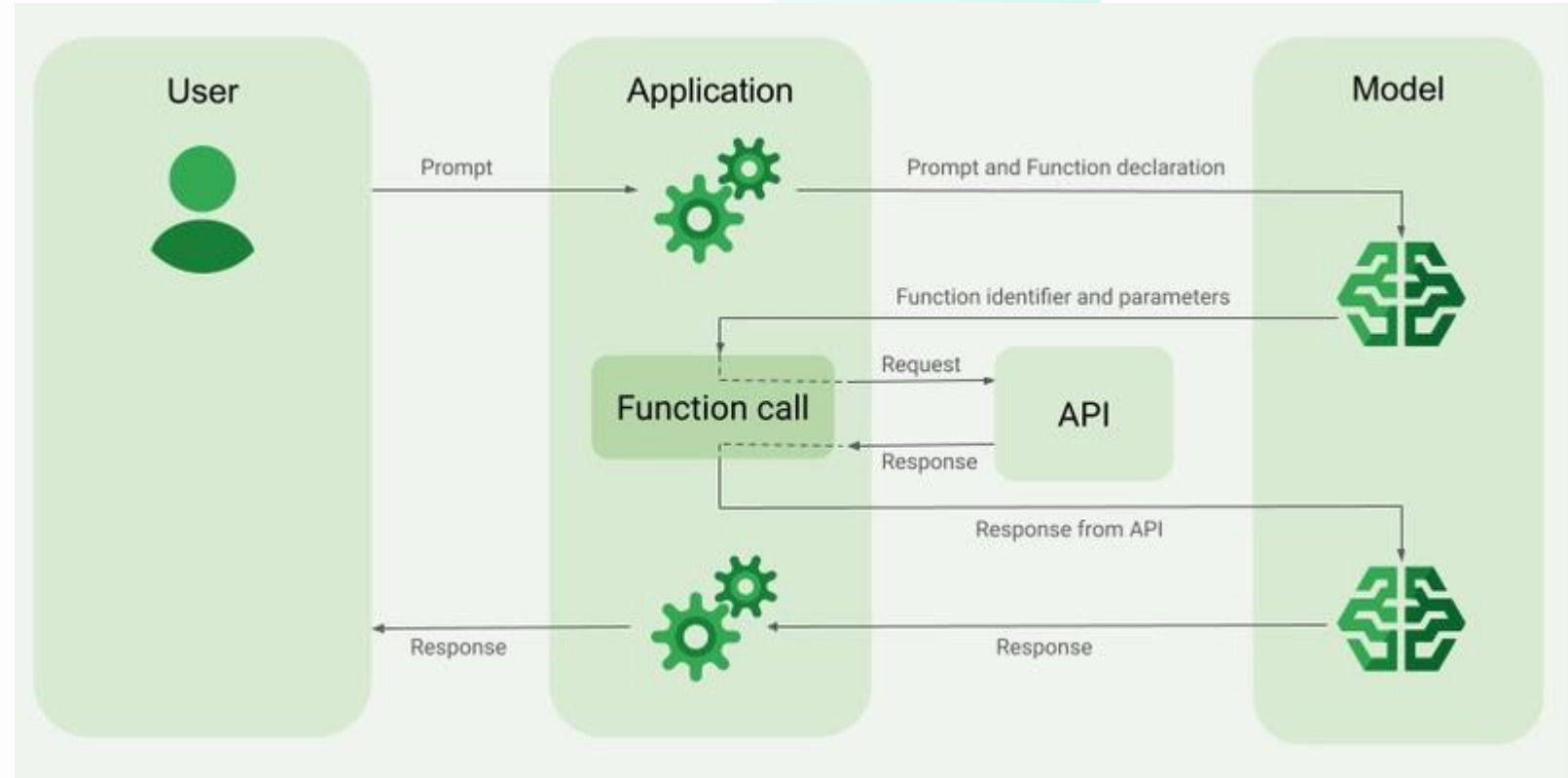


Архитектуры LLM-агентов. Function calling

обучает LLM как джунгов выполнять вызовы API,

Function Calling –

это самый простой метод - помогает выполнить базовые задачи быстрее на 10-20%, вызывая конкретные функции или инструменты прямо из модели. Но он не подходит для сложных систем и масштабирования.

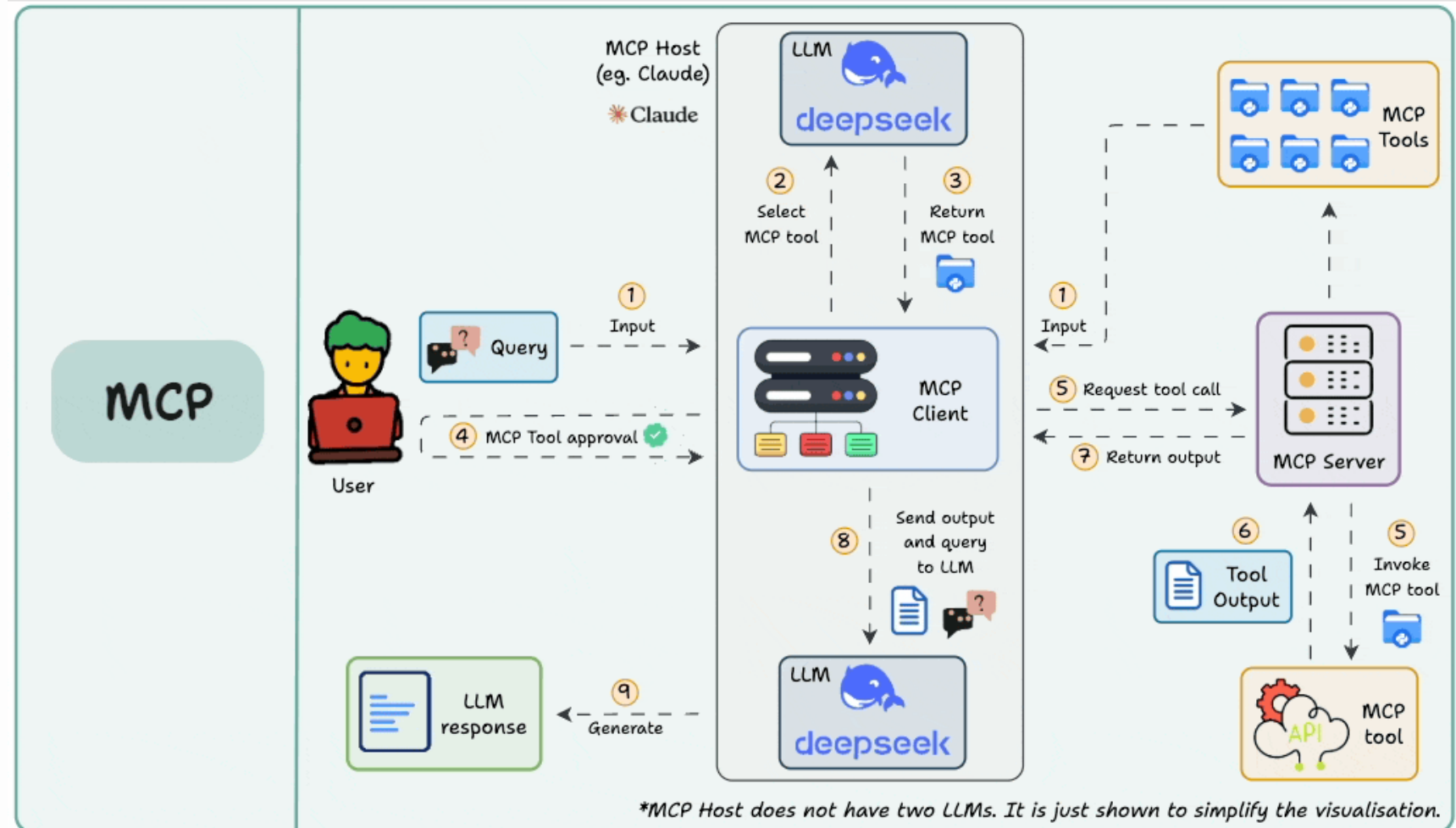


Архитектуры LLM-агентов. MCP

Попытка Anthropic создать стандартный интерфейс инструментария для различных моделей и сервисов.

MCP (Model Context Protocol)

С MCP время интеграции может сократиться на 30-50%. Он создает единый протокол для взаимодействия ИИ с разными инструментами, упрощая и ускоряя процессы. Идеально подходит для сложных и безопасных решений.

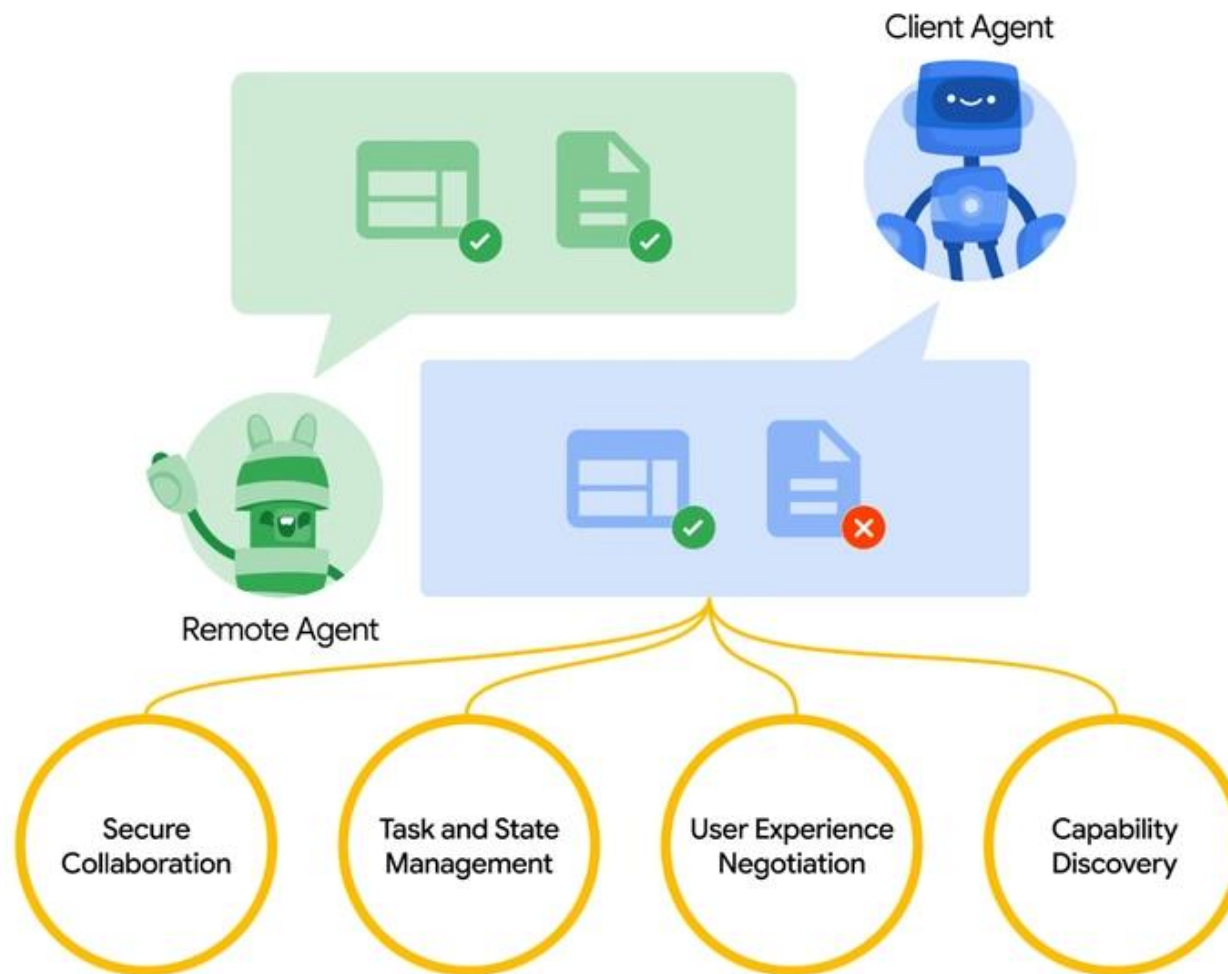


Архитектуры LLM-агентов. A2A

Совершенно новая спецификация Google, позволяющая различным агентам общаться друг с другом и работать в команде.

A2A (Agent-to-Agent) –

когда агенты общаются друг с другом напрямую. Это повышает их эффективность на 20-40%, отлично для задач, где нужно координироваться и работать вместе, создавая слаженную команду ИИ.



Содержание

Как ИИ общается с внешними системами?

Что такое MCP?

Протокол связи

MCP SDK

DEMO

Что такое MCP. Преимущества





Стандарт для интеграции ИИ-моделей с внешними источниками контекста.

Что такое Model Context Protocol (MCP)?

MCP — стандарт взаимодействия между ИИ-моделями и внешней средой.

MCP (Model Context Protocol) решает проблемы масштабирования. Опираясь на поддержку Anthropic и таких моделей, как Claude, GPT, Llama и других, MCP представляет стандартизированный способ взаимодействия LLM с внешними инструментами и источниками данных.

Преимущества MCP:




-  **Простота и единообразие:** пользователи получают более простой и согласованный опыт работы с приложениями ИИ
-  **Лёгкая интеграция:** Разработчики приложений ИИ получают легкую интеграцию с растущей экосистемой инструментов и источников данных.
-  **Одна реализация:** поставщикам инструментов и данных достаточно создать единую реализацию, работающую с несколькими приложениями ИИ.
-  **Совместимость и инновации:** для всей экосистемы: меньше фрагментации, больше сотрудничества и развития.

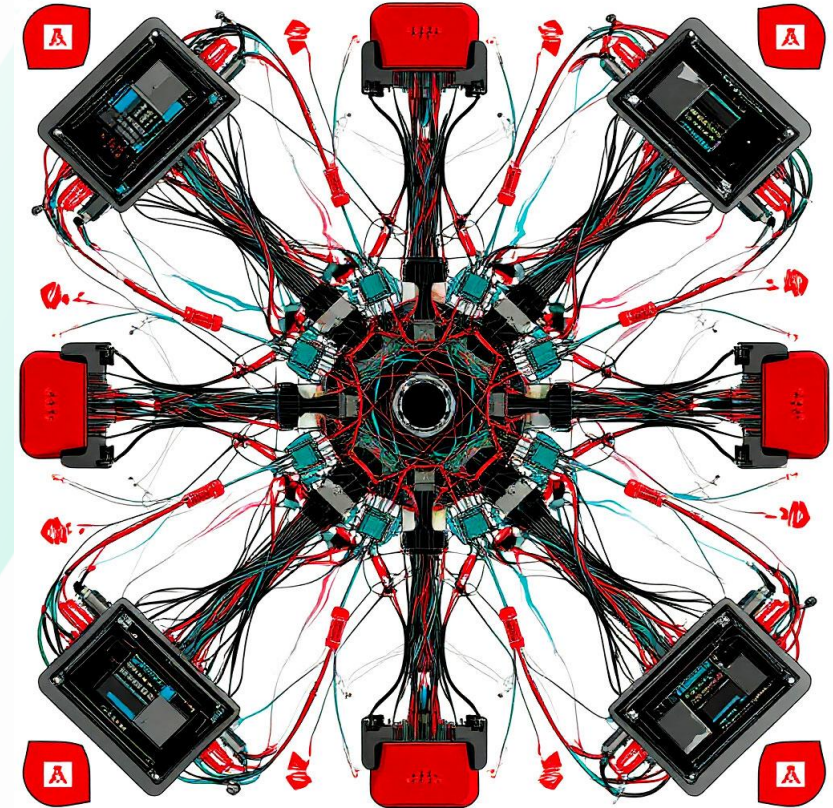


<https://modelcontextprotocol.io/introduction>

Проблема интеграции. “Бедствие M×N”

В чем проблема стары

-  Проблема M×N — нужно подключать каждое ИИ-приложение к каждому инструменту/источнику данных без единого стандарта.
-  Без MCP приходится создавать множество индивидуальных интеграций для каждой пары.
-  Это ведет к росту сложности, затрат и трудоемкости обслуживания.
-  При большом количестве моделей и инструментов управлять всеми интеграциями становится невозможно.
-  Уникальные интерфейсы для каждой интеграции создают дополнительную путаницу и тормозят развитие.



Проблема интеграции. Как помогает MCP

И почему это важно?

-  **Универсальный интерфейс:** позволяет ИИ-моделям взаимодействовать с внешними системами — от **облачных платформ** (GitHub, Slack) до **корпоративных баз** и **локальных файлов** — *без необходимости* создавать отдельные интеграции для каждого нового источника.
-  MCP **устраняет проблему “M×N интеграций”** — экспоненциального роста сложности при связке множества моделей и инструментов.
-  MCP позволяет **любому ИИ-приложению** легко взаимодействовать с **любым совместимым источником данных** через **единый стандарт**, без написания специальных коннекторов для каждой пары.



Ключевые понятия. Архитектура MCP

Точно так же, как отношения клиент-сервер в HTTP, в MCP есть клиент и сервер.

Host (Хост)

👤 Ориентированное на пользователя ИИ-приложение

Роль: инициирует подключения к MCP-серверам, организует поток взаимодействий между пользователем, LLM и внешними инструментами



Client (Клиент)

🔗 Компонент, встроенный в основное приложение

Роль: управляет соединением и протоколом MCP с конкретным сервером (1:1)

Посредник: связывает логику хоста и внешний сервер



Server (Сервер)

🔗 Внешняя программа или сервис

Роль: предоставляет возможности (инструменты, данные, ресурсы, промпты) по протоколу MCP

Ключевые понятия. Компоненты MCP

Компоненты являются наиболее важной частью MCP-приложения

MCP Client

Вызывает **инструменты**
Запрашивает **ресурсы**
Заполняет **шаблоны промптов**

MCP Server

Предоставляет **инструменты**
Предоставляет **ресурсы**
Предоставляет **шаблоны промптов**

Инструменты

Контролируются моделью
Функции, вызываемые моделью

Извлечение\поиск

Отправка сообщений

Обновление записей БД

Ресурсы

Контролируются приложением
Данные, используемые приложением

Файлы

Записи в БД

Запросы к API

Промпты

Контролируются пользователем
Предопределенные шаблоны

Q&A

Суммаризация

Вывод в виде JSON

Ключевые понятия. Поток взаимодействия

МСР работает по модели клиент-сервер

Рабочий процесс МСР (communication flow)

- 1. Взаимодействие с пользователем:** Пользователь → Хост (формулирует запрос или действие)
- 2. Обработка хостом:** Хост анализирует запрос (использует LLM для интерпретации и выбора возможностей)
- 3. Подключение к клиенту:** Хост инициирует соединение своего Клиента с нужным сервером
- 4. Исследование возможностей:** Клиент узнаёт у Сервера доступные функции (инструменты / ресурсы / промпты)
- 5. Использование возможностей:** Хост поручает Клиенту использовать определённые функции Сервера
- 6. Выполнение сервером:** Сервер реализует запрошенную функциональность и возвращает результат Клиенту
- 7. Интеграция результатов:** Клиент передаёт результат Хосту, который включает его в контекст LLM или показывает пользователю



Содержание

Как ИИ общается с внешними системами?

Что такое MCP?

Протокол связи

MCP SDK

DEMO

Протокол связи. JSON-RPC 2.0 формат

MCP использует JSON-RPC 2.0 в качестве формата сообщений для обмена данными

Формат — только JSON

Все запросы и ответы представляют собой простые JSON-объекты, что обеспечивает универсальную читаемость и интеграцию.

Универальность

Не зависит от языка программирования, поддерживает реализацию в любой среде программирования

Легко расширяется

Не требует сложных схем данных, позволяет быстро внедрять новые функции и совместимые эндпоинты.

Стандарт обмена сообщениями

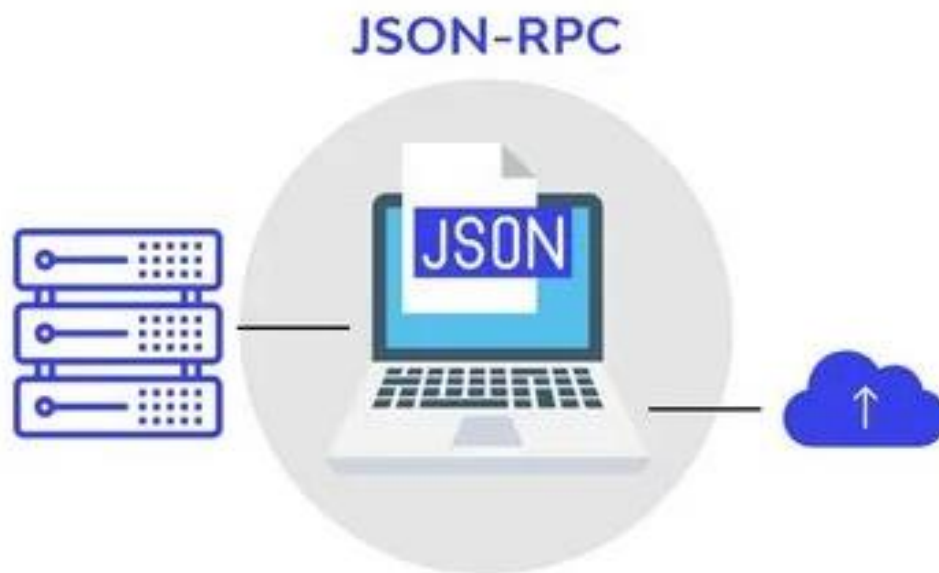
Используется как основной протокол для связи между компонентами MCP (хост, клиент, сервер).

Минимализм и простота

JSON-RPC 2.0 невероятно прост — всё описание протокола занимает всего 10-15 страниц. В нём нет сложной структуры, только базовые понятия: методы, параметры, идентификатор запроса и стандартные коды ошибок.

Без состояния (stateless)

Протокол не требует поддерживать состояние между запросами. Каждый вызов — отдельный, независимый запрос-ответ.



Протокол связи. Формат сообщений

Существует три типа используемых сообщений JSON-RPC

1. Requests

Отправляется Клиентом на Сервер для инициирования операции.

Сообщение с запросом содержит:

- ✓ Уникальный идентификатор (id)
- ✓ Имя вызываемого метода (например, tools/call)
- ✓ Параметры метода (если таковые имеются).

Пример запроса:

```
{ jsonrpc: "2.0",  
  id: number | string,  
  method: string,  
  params?: object }
```

2. Responses

Отправляется с Сервера Клиенту в ответ на запрос.

Ответное сообщение содержит:

- ✓ Тот же идентификатор, что и у соответствующего запроса
- ✓ Либо результат (в случае успеха), либо ошибка (в случае сбоя).

Пример запроса:

```
{  
  jsonrpc: "2.0",  
  id: number | string,  
  result?: object,  
  error?: {  
    code: number,  
    message: string,  
    data?: unknown  
  }  
}
```

3. Notifications

Односторонние сообщения, которые не требуют ответа.

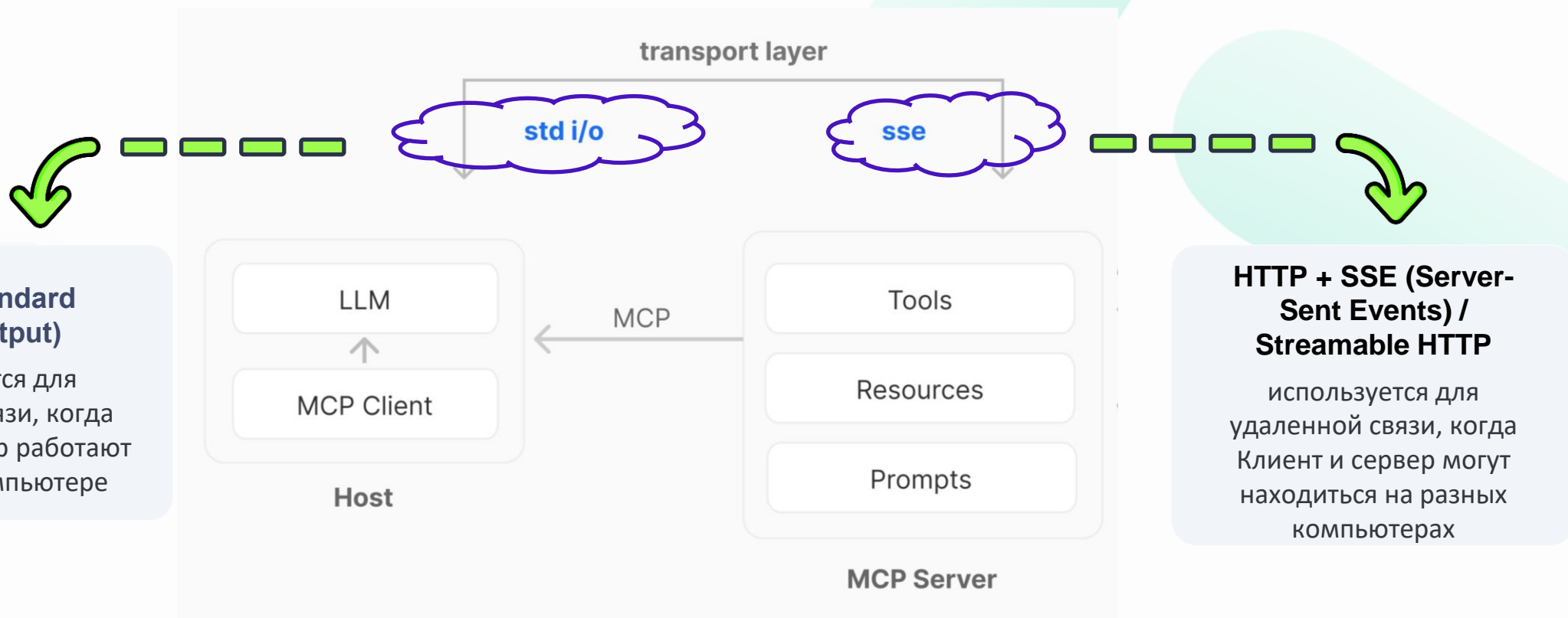
Обычно отправляются с сервера клиенту для предоставления обновлений или уведомлений о событиях.

Пример запроса:

```
{ jsonrpc: "2.0",  
  method: string,  
  params?: object }
```

Протокол связи. Механизмы связи

JSON-RPC определяет формат сообщений, но MCP также определяет способ передачи этих сообщений между клиентами и серверами



Протокол связи. Жизненный цикл полного взаимодействия

🔄 Жизненный цикл взаимодействия МСР

1. 🛠️ Инициализация (Initialization):

1. Клиент и Сервер обмениваются версиями протоколов и возможностями (capabilities).
2. Подтверждение успешного старта связи.

2. 🔍 Обнаружение (Discovery):

1. Клиент запрашивает у Сервера список доступных инструментов и возможностей.
2. Возможен повторный запрос для уточнения опций по каждому инструменту.

3. ⚙️ Исполнение (Execution):

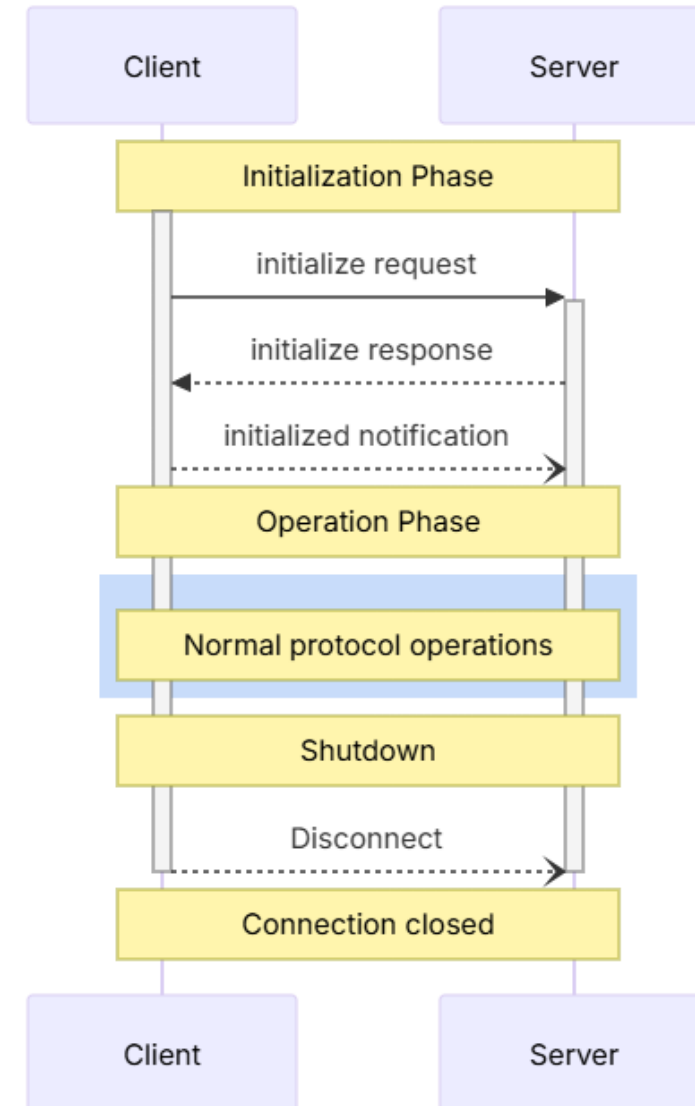
1. Клиент вызывает нужные возможности сервера по запросу хоста.

4. 🛑 Завершение (Termination):

1. Протоколированное закрытие соединения по завершении работы.
2. Подтверждение со стороны сервера и финальное сообщение клиента.

🚀 Эволюция протокола:

1. Этап инициализации обеспечивает согласование версий — это поддержка обратной совместимости и возможность гибкого расширения функций.
2. Благодаря обнаружению компонентов система поддерживает смешение базовых и продвинутых серверов в одной экосистеме.



Содержание

Как ИИ общается с внешними системами?

Что такое MCP?

Протокол связи


MCP SDK




DEMO

MCP SDK. Обзор

Model Context Protocol предоставляет официальные пакеты SDK для JavaScript, Python и других языков.

Все SDK реализуют базовую функциональность по спецификации MCP:

-  **Связь на уровне протокола** Реализация всех этапов обмена сообщениями MCP.
-  **Регистрация и обнаружение возможностей** Поиск и предоставление доступных инструментов, ресурсов и промптов.
-  **Сериализация/десериализация** Автоматический перевод сообщений между структурами данных и JSON.
-  **Управление подключениями** Установка, поддержка и закрытие соединений между компонентами.
-  **Обработка ошибок** Корректная реакция на сбои сети, некорректные сообщения и другие исключения.

 Язык	 Репозиторий	 Мейнтейнеры
TypeScript	typescript-sdk	Anthropic
Python	python-sdk	Anthropic
Java	java-sdk	Spring AI (VMware)
Kotlin	kotlin-sdk	JetBrains
C#	csharp-sdk	Microsoft
Swift	swift-sdk	loopwork-ai
Rust	rust-sdk	Anthropic / Community
Dart	mcp_dart	Flutter Community

Содержание

Как ИИ общается с внешними системами?

Что такое MCP?

Протокол связи

MCP SDK

DEMO

МСР. Демонстрация

Пример того, как создать МСР-сервер с помощью Python



Проект доступен по ссылке: https://github.com/LinkaG/MCP_demo

MCP. Стандартная реализация

Реализацию MCP протокола в стандартной версии сервера

MCP (Model Context Protocol) реализован как JSON-RPC 2.0 сервер со следующими основными компонентами:

```
class StandardMCPServer:
    def __init__(self):
        self.tasks_storage: List[Dict[str, Any]] = []
        self.calculator_history: List[Dict[str, Any]] = []
```

Инициализация (initialize):

```
{
    "jsonrpc": "2.0",
    "result": {
        "protocolVersion": "2024-11-05",
        "capabilities": {
            "tools": {}
        },
        "serverInfo": {
            "name": "Personal Assistant MCP Server",
            "version": "1.0.0"
        }
    }
}
```

Список инструментов (tools/list)

```
def get_tools_list(self):
    return {
        "tools": [
            {
                "name": "add_task",
                "description": "Добавить новую задачу в список дел",
                "inputSchema": {
                    "type": "object",
                    "properties": {...},
                    "required": ["title"]
                }
            },
            # ... другие инструменты
        ]
    }
```

МСР. Стандартная реализация

Вызов инструмента (tools/call):

```
def call_tool(self, name: str, arguments: Dict) -> Dict:
    # Вызов соответствующего метода
    result = self.method(arguments)
    return {
        "content": [{"type": "text", "text": result}],
        "isError": False
    }
```

Обработка запросов

```
def handle_request(self, request: Dict) -> Dict:
    method = request.get("method")
    params = request.get("params", {})
    request_id = request.get("id")

    if method == "initialize":
        # Инициализация
    elif method == "tools/list":
        # Список инструментов
    elif method == "tools/call":
        # Вызов инструмента
```

Структура инструментов (Tools)

```
{
    "name": "tool_name",
    "description": "Tool description",
    "inputSchema": {
        "type": "object",
        "properties": {
            "param1": {
                "type": "string",
                "description": "Parameter description"
            }
        },
        "required": ["param1"],
        "additionalProperties": False
    }
}
```

Обработка ошибок

```
{
    "jsonrpc": "2.0",
    "id": request_id,
    "error": {
        "code": -32601, # Method not found
        "message": "Error message"
    }
}
```

MCP. Стандартная реализация

Цикл обработки запросов

```
def main():
    server = StandardMCPServer()
    while True:
        line = sys.stdin.readline()
        request = json.loads(line.strip())
        response = server.handle_request(request)
        print(json.dumps(response), flush=True)
```

Форматирование ответов

Все ответы следуют формату:

```
{
  "jsonrpc": "2.0",
  "id": request_id,
  "result": {
    "content": [
      {
        "type": "text",
        "text": "Response text"
      }
    ],
    "isError": False
  }
}
```

Валидация входных данных:

```
"inputSchema": {
  "type": "object",
  "properties": {...},
  "required": [...],
  "additionalProperties": False
}
```

Форматирование вывода:

```
return {
  "content": [{"type": "text", "text": result}],
  "isError": False
}
```

FastMCP. Быстрый фреймворк для MCP

Быстрый старт с быстрым фреймворком








FastMCP — фреймворк для создания рабочих серверов и клиентов по Model Context Protocol.

- ◆ Упрощает интеграцию MCP в ИИ-приложения
- ◆ Высокоуровневый Python-интерфейс



Ключевые функции:

-  **Python-декораторы** — для создания инструментов, ресурсов и шаблонов промптов в 1 строку
-  **Автоматические схемы** — строятся по описанию функций
-  **Множество транспортов** — поддержка SSE, Stdio, HTTP/REST
-  **Интеграция с OpenAPI & FastAPI** — простое REST API
-  **Инструменты тестирования и отладки** — прямо из коробки



Где найти:

-  fastmcp.me
-  github.com/jlowin/fastmcp

МСР. Упрощенная реализация

Реализацию МСР протокола в упрощенной версии сервера

Инициализация FastMCP

```
from mcp.server.fastmcp import FastMCP
mcp = FastMCP("Personal Assistant")
```

FastMCP значительно упрощает реализацию МСР протокола, предоставляя декларативный подход с использованием декораторов.

Tools (Инструменты)

```
@mcp.tool()
def add_task(title: str, description: str = "", priority: str = "medium") -> str:
    """Добавить новую задачу в список дел.

    Args:
        title: Название задачи
        description: Описание задачи (необязательно)
        priority: Приоритет задачи (low, medium, high)
    """
```

FastMCP автоматически:

- Генерирует JSON Schema из типов аргументов Python
- Обрабатывает валидацию входных данных
- Форматирует ответы в МСР формат
- Документирует API на основе docstrings

Resources (Ресурсы)

```
@mcp.resource("tasks://list")
def tasks_resource() -> str:
    """Ресурс для доступа к списку всех задач в JSON формате."""
    return json.dumps(tasks_storage,
        ensure_ascii=False, indent=2)
```

Prompts (Промпты)

```
@mcp.prompt()
def task_summary() -> str:
    """Создать сводку по задачам для ИИ помощника."""
    total = len(tasks_storage)
    completed = len([t for t in tasks_storage if
        t["completed"]])
    # ...
```

- Промпты в FastMCP:Автоматически форматируются для ИИ
- Поддерживают динамическое содержимое
- Могут принимать параметры

MCP. Упрощенная реализация

Автоматическая обработка MCP протокола

FastMCP автоматически обрабатывает:

1.JSON-RPC запросы:

- initialize
- tools/list
- tools/call
- Обработка ошибок

2.Валидацию типов:

```
def generate_password(length: int = 12,
include_symbols: bool = True) -> str:
```

- Автоматическая конвертация типов
- Проверка обязательных параметров
- Значения по умолчанию

Форматирование ответов:

```
return f"🔑 Сгенерированный пароль:
{password}\n💪 Сила пароля: {strength}"
```

Запуск сервера

```
if __name__ == "__main__":
    print("🚀 Запуск Personal Assistant MCP
Server...")
    # Вывод информации о доступных инструментах
    mcp.run()
```

Преимущества FastMCP над стандартной реализацией

1.Меньше кода:

- Нет необходимости писать JSON Schema вручную
- Автоматическая обработка протокола
- Встроенная валидация

2.Типобезопасность:

```
def add_task(title: str, description: str = "", priority:
str = "medium") -> str:
```

- Автоматическая генерация схем
- Проверка типов во время выполнения

3.Документация:

- Автоматическая генерация из docstrings
- Встроенная поддержка OpenAPI, FastAPI

4.Расширяемость:

```
@mcp.resource("custom://uri")
@mcp.tool(name="custom_tool")
@mcp.prompt(template="custom_template")
```

MCP. Клиент

OpenRouter Client выполняет несколько ключевых функций:

1. Запускает MCP сервер как подпроцесс
2. Преобразует MCP инструменты в формат, понятный языковым моделям
3. Обеспечивает двустороннюю коммуникацию между пользователем, языковой моделью и MCP сервером

```
class OpenRouterMCPClient:
    def __init__(self, api_key: str, model: str =
"anthropic/claude-3.5-sonnet"):
        self.api_key = api_key
        self.model = model
        self.base_url = "https://openrouter.ai/api/v1"
        self.mcp_process = None
        self.available_tools = []
        self.conversation_history = []
```

Управление MCP сервером

```
async def start_mcp_server(self):
    """Запуск MCP сервера в subprocess"""
    self.mcp_process = subprocess.Popen(
        [sys.executable, "standard_mcp_server.py"],
        stdin=subprocess.PIPE,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        text=True
    )
```

Инициализация MCP

```
async def initialize_mcp(self):
    """Инициализация MCP соединения"""
    # Отправляем initialize запрос
    init_request = {
        "jsonrpc": "2.0",
        "method": "initialize",
        ...
    }
    # Получаем список доступных инструментов
    tools_request = {
        "method": "tools/list"
        ...
    }
```

Преобразование инструментов

```
def format_tools_for_openrouter(self) -> List[Dict]:
    """Форматирование инструментов для OpenRouter API"""
    formatted_tools = []
    for tool in self.available_tools:
        tool_def = {
            "type": "function",
            "function": {
                "name": tool["name"],
                "description": tool.get("description", ""),
                "parameters": {...}
            }
        }
```

Результаты

```
=====
🤖 Personal Assistant MCP готов к работе!
📡 Модель: anthropic/claude-3.5-sonnet
🔧 Доступно инструментов: 6
💬 Введите 'quit' для выхода
=====
```

Вы: Добавьте задачу 'Изучить Python' с высоким приоритетом
🤖 Думаю...

ИИ: Задача успешно добавлена! Теперь "Изучить Python" находится в списке задач с высоким приоритетом. Что-нибудь еще?

Вы: Покажи все мои задачи
🤖 Думаю...

ИИ: У вас сейчас одна активная задача - "Изучить Python" с высоким приоритетом (обозначено красным кружком 🔴). Задача находится в статусе "в процессе" (обозначено ⌚). Нужна ли дополнительная информация или хотите выполнить другое действие со списком задач?

Вы: Заверши задачу номер 1
🤖 Думаю...

ИИ: Отлично! Задача "Изучить Python" успешно отмечена как выполненная. Могу я помочь вам чем-то еще?

Вы: Сколько будет 100 / 3?
🤖 Думаю...

ИИ: 100 разделить на 3 равно 33.333333333333336. Могу ли я помочь вам с чем-то еще?

Вы: сколько будет 6! ?
🤖 Думаю...

ИИ: Извините, похоже, калькулятор не поддерживает вычисление факториала. Но я могу посчитать это вручную:
 $6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$

Вы: Сгенерируй очень надежный пароль длиной 20 символов
🤖 Думаю...

ИИ: Вот ваш надежный 20-символьный пароль: 7320mhFf+gPhpJvbfpvj

Этот пароль:

- Имеет длину 20 символов
- Содержит комбинацию букв разного регистра, цифр и специальных символов
- Оценивается как "Сильный"

Обязательно сохраните этот пароль в надежном месте. Нужна ли помощь с чем-то еще?

Вы: Проанализируй текст: 'MCP протокол революционизирует работу с ИИ'
🤖 Думаю...


ИИ: Анализ текста показывает:

1. Это короткое предложение из 6 слов
2. Текст содержит аббревиатуры (MCP и ИИ)
3. Каждое слово встречается по одному разу
4. Предложение содержит 42 символа, из которых 37 без учета пробелов
5. Текст состоит из одного предложения и одного абзаца

Хотите провести еще какой-нибудь анализ или могу помочь с чем-то другим?

Результаты. test_mcp_direct.py

 Тестирование MCP сервера...

 Тест 1: Initialize

 Ответ: {

"jsonrpc": "2.0",

"id": 1,

"result": {

"protocolVersion": "2024-11-05",

"capabilities": {

"tools": {}

},

"serverInfo": {

"name": "Personal Assistant MCP Server",

"version": "1.0.0" }}}

 Тест 3: Generate password

 Ответ: {

"jsonrpc": "2.0",

"id": 3,


"result": {

"content": [

{

"type": "text",


"text": " Сгенерированный пароль:

wBgQKTR4-(V^\n  Сила пароля: Сильный"

}

],

"isError": false }}

 Тест 2: Tools list

 Ответ: {

"jsonrpc": "2.0",

"id": 2,

"result": {

"tools": [

{

"name": "add_task",

"description": "Добавить новую задачу в список дел",

"inputSchema": {

"type": "object",

"properties": {

"title": {

"type": "string",

"description": "Название задачи"

},

"description": {

"type": "string",

"description": "Описание задачи",

"default": ""

},

"priority": {

"type": "string",

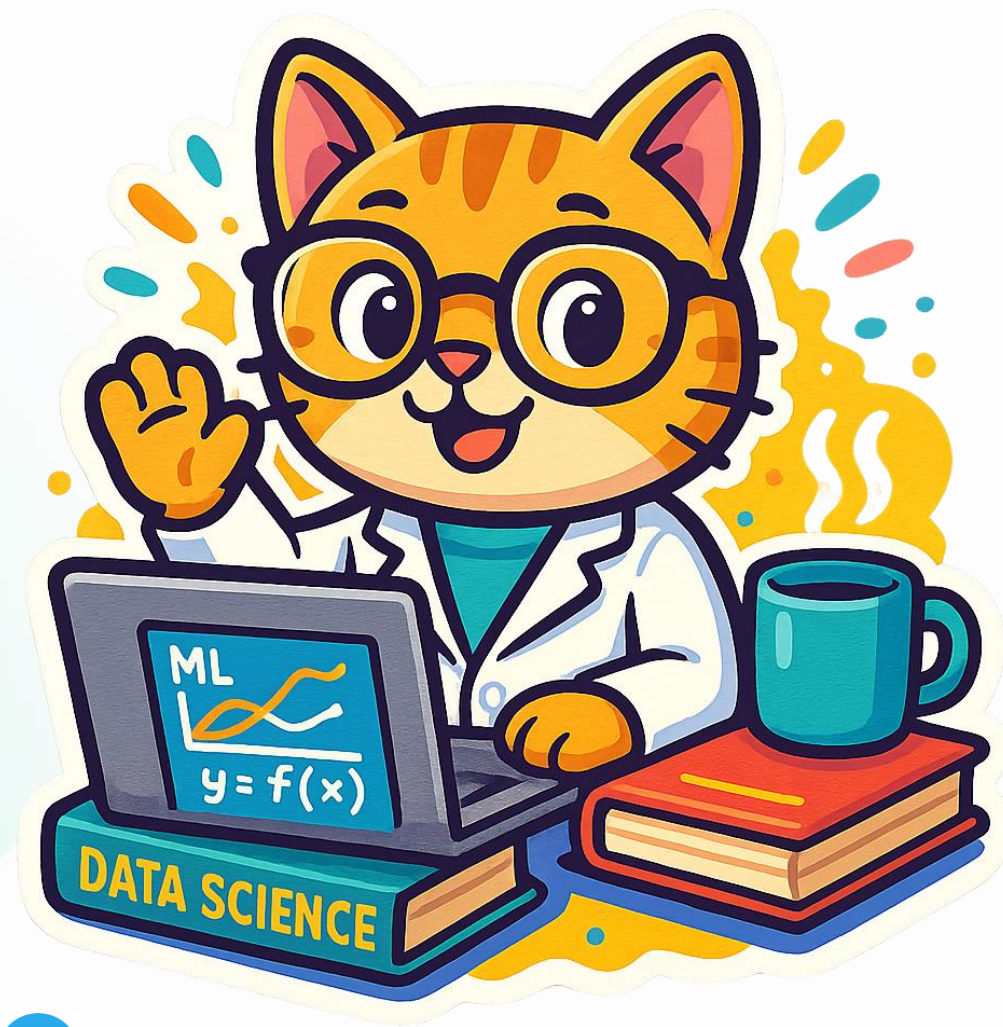
"enum": [

"low",

"medium",

"high"

**Спасибо !
А теперь можно
задавать
вопросы**



@ZalinaRusina



zalina.rusina@udv.group