

Автоматизация скриптов для системного администрирования с использованием мультимодальной модели GPT-5.2

Саяпин Никита Анатольевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Аннотация

В статье рассматривается процесс автоматической генерации скриптов для типичных задач системного администрирования (bash и PowerShell) на основе текстового описания с применением мультимодальной нейросети GPT-5.2. Показано, как модель интерпретирует требования, генерирует корректный и безопасный код, а также анализируется влияние уровня детализации промпта на качество скриптов и их применимость в реальных сценариях.

Ключевые слова: GPT-5.2, генерация скриптов, системное администрирование, автоматизация задач, bash, PowerShell, мультимодальные модели.

Generation of scripts for system administration using the GPT-5.2 multimodal model

Sayapin Nikita Anatolyevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article discusses the process of automatic generation of scripts for typical system administration tasks (bash and PowerShell) based on a text description using the GPT-5.2 multimodal neural network. It is shown how the model interprets requirements, generates correct and safe code, and analyzes the impact of prompt detail level on script quality and applicability in real scenarios.

Keywords: GPT-5.2, script generation, system administration, task automation, bash, PowerShell, multimodal models.

1. Введение

1.1 Актуальность

Системные администраторы ежедневно сталкиваются с повторяющимися задачами: резервное копирование данных, мониторинг дискового пространства, очистка временных файлов, управление пользователями и службами. Написание скриптов для автоматизации этих процессов требует времени, знаний синтаксиса bash или PowerShell и внимания к безопасности (избежание уязвимостей, обработка ошибок). В

условиях дефицита времени и роста сложности инфраструктуры традиционный подход становится неэффективным.

Мультимодальные модели нового поколения, такие как GPT-5.2, позволяют автоматизировать создание скриптов по естественному языковому описанию задачи. Модель может генерировать код, добавлять комментарии, учитывать лучшие практики (безопасность, логирование, параметры) и предлагать варианты для разных ОС. Это ускоряет работу администратора, снижает вероятность ошибок и делает автоматизацию доступной даже для начинающих специалистов. Актуальность особенно высока в эпоху облачных инфраструктур, DevOps и SRE, где скорость и надёжность автоматизации критически важны.

1.2 Обзор исследований

А. О. Суворов и В. А. Суворова рассмотрели применение методов интеллектуального анализа сетевого трафика для построения систем обнаружения вторжений, показав, что использование искусственных нейронных сетей позволяет повысить качество идентификации сетевых атак за счёт автоматизированного выделения информативных признаков в потоках данных [1]. А. М. Митрахович выполнил сравнительный анализ алгоритмов искусственного интеллекта, применяемых для защиты компьютерных сетей, и продемонстрировал, что эффективность различных методов машинного обучения существенно зависит от типа атак и характеристик контролируемой сети [2]. И. Б. Саенко исследовал эволюцию подходов к искусственному интеллекту и современное состояние исследований в этой области, уделив особое внимание задачам информационно-телекоммуникационных систем и отметив тенденцию перехода от локальных экспертных решений к комплексным интеллектуальным платформам управления инфраструктурой [3]. Е. Долматов проанализировал подходы к применению технологий искусственного интеллекта для решения задач автоматизированного управления в сложных технических системах, обосновав целесообразность использования ИИ-моделей для поддержки принятия решений и повышения степени автономности таких систем [4]. В. В. Аллакин предложил общий подход к построению перспективных систем мониторинга технических объектов, включая сетевое оборудование, показав, что интеллектуальная обработка телеметрии и автоматическая генерация предупреждений и отчётов позволяют повысить надёжность функционирования распределённой инфраструктуры [5].

1.3 Цель исследования

Целью исследования является демонстрация возможностей мультимодальной нейросети GPT-5.2 в генерации скриптов для системного администрирования на основе текстового описания задач, а также оценка качества получаемого кода с точки зрения корректности, безопасности, читаемости и зависимости результата от детализации промпта.

2 Материалы и методы

В исследовании применяется мультимодальная модель GPT-5.2, которая способна воспринимать текстовые описания задач системного администрирования и генерировать на их основе исполняемый код на языках `bash` и `PowerShell`. Модель рассматривается как инструмент преобразования естественного языка в готовые скрипты без необходимости предварительной подготовки данных.

Подготавливаются текстовые промпты, различающиеся по степени детализации: от кратких формулировок, содержащих только основную цель задачи, до среднетализированных, где указывается предпочитаемый язык программирования, основные требования и необходимость обработки ошибок, и далее до высоко детализированных, включающих полное техническое задание с описанием параметров, логирования, мер безопасности и дополнительных проверок.

Выбираются типичные административные сценарии: создание резервных копий директорий, мониторинг свободного места на дисках, автоматическая очистка временных файлов, проверка состояния служб и аналогичные рутинные операции. Генерируемые скрипты подвергаются анализу по нескольким критериям: проверяется синтаксическая правильность кода, оценивается полнота выполнения поставленной задачи, анализируется уровень безопасности (отсутствие опасных конструкций, корректная обработка входных данных), изучается читаемость (наличие комментариев, логичная структура, понятные имена переменных). Особое внимание уделяется тому, как степень детализации исходного промпта влияет на итоговое качество скрипта.

Оценка проводится путём визуального анализа сгенерированного кода и мысленного моделирования его выполнения в типичных условиях работы системного администратора.

3. Результаты

3.1 Генерация простого скрипта для резервного копирования по минимальному описанию

В первом сценарии тестируется базовая способность модели создавать скрипт на основе крайне краткого промпта, где указана только суть задачи без дополнительных уточнений. Это имитирует ситуацию, когда администратор быстро формулирует идею, не задумываясь о деталях, и ожидает от ИИ "умного" заполнения пробелов.

Промпт:

«Создай скрипт для резервного копирования директории `/var/log` в архив.»

GPT-5.2 самостоятельно выбрала язык `bash` (как наиболее распространённый для Unix-подобных систем) и сгенерировала лаконичный, но функциональный код. Скрипт использует команду `tar` для сжатия, добавляет метку времени в имя файла для уникальности и выводит простое сообщение об успехе. Интересно, что модель автоматически учла

потенциальные риски, добавив опцию -P для сохранения абсолютных путей, что делает скрипт более надёжным для системных директорий (Рис. 1).

```
#!/bin/bash
timestamp=$(date +%Y%m%d_%H%M%S)
tar -czPf /backup/logs_backup_${timestamp}.tar.gz /var/log
if [ $? -eq 0 ]; then
    echo "Резервная копия /var/log успешно создана: logs_backup_${timestamp}.tar.gz"
else
    echo "Ошибка при создании резервной копии"
fi
```

Рисунок 1 — Bash скрипт для резервного копирования

Этот скрипт прост в использовании и идеален для новичков, но без обработки сложных ошибок (например, нехватки места на диске) он подходит только для тестовых окружений. В реальной практике такой подход экономит время на рутинные задачи, позволяя администратору сосредоточиться на более сложных аспектах инфраструктуры.

3.2 Генерация скрипта для мониторинга ресурсов с учётом базовых параметров

На втором этапе промпт усложняется добавлением параметров и условий, что позволяет оценить, как модель интегрирует логику принятия решений. Это делает процесс интересным, поскольку демонстрирует "интеллект" ИИ в предвидении типичных административных нужд, таких как уведомления о критических состояниях.

Промпт:

«Напиши PowerShell-скрипт для мониторинга CPU и памяти на сервере. Если CPU > 80% или память < 20% свободной — запиши в лог и выведи предупреждение.»

Модель создала скрипт, использующий встроенные cmdlet PowerShell для сбора метрик (Get-WmiObject для CPU и Get-CimInstance для памяти), с расчётом процентов и условными проверками. Интересный аспект: GPT-5.2 добавила цикл для периодического мониторинга (каждые 60 секунд), хотя это не было явно указано, что превращает скрипт в мини-мониторинг-тул (Рис. 2).

```
while ($true) {  
    $cpuLoad = (Get-WmiObject Win32_Processor | Measure-Object -Property LoadPercentage -Average).Average  
    $mem = Get-CimInstance Win32_OperatingSystem  
    $freeMemPercent = [math]::Round(($mem.FreePhysicalMemory / $mem.TotalVisibleMemorySize) * 100, 2)  
  
    if ($cpuLoad -gt 80 -or $freeMemPercent -lt 20) {  
        $msg = "$(Get-Date): Критическое состояние! CPU: $cpuLoad%, Свободная память: $freeMemPercent%"  
        Write-Warning $msg  
        Add-Content -Path "C:\Logs\ServerMonitor.log" -Value $msg  
    } else {  
        Write-Output "$(Get-Date): Система в норме (CPU: $cpuLoad%, Память: $freeMemPercent%)"  
    }  
    Start-Sleep -Seconds 60  
}
```

Рисунок 2 — Файл скрипта мониторинга с примерами предупреждений

Этот скрипт не только выполняет задачу, но и добавляет элемент "проактивности", делая его полезным для реального мониторинга в облачных или виртуальных средах. Без остановки он может работать как демон, что открывает двери для интеграции с системами вроде Task Scheduler

3.3 Генерация скрипта для управления пользователями с элементами безопасности

Третий запрос фокусируется на задачах, связанных с пользователями, где безопасность играет ключевую роль. Здесь интересно наблюдать, как модель балансирует между простотой и защитой от ошибок, автоматически включая проверки прав и хэширование паролей.

Промпт:

«Сгенерируй bash-скрипт для добавления нового пользователя с паролем и назначением группы. Учти проверку на существование пользователя и логирование.»

GPT-5.2 создала скрипт с использованием `useradd`, `passwd` и `id`, добавив опции для безопасного ввода пароля (без эха) и проверки. Модель учла лучшие практики: запуск только от `root`, логирование в `syslog` и обработку ошибок. Код получился компактным, но надёжным (Рис. 3).

```
if [ "$(id -u)" != "0" ]; then
    echo "Скрипт должен запускаться от root" 1>&2
    exit 1
fi

read -p "Имя нового пользователя: " username
if id "$username" >/dev/null 2>&1; then
    echo "Пользователь $username уже существует"
    exit 1
fi

read -s -p "Пароль: " password
echo

useradd -m -G users $username
echo "$username:$password" | chpasswd

if [ $? -eq 0 ]; then
    logger -t admin_script "Добавлен пользователь $username"
    echo "Пользователь $username успешно добавлен"
else
    echo "Ошибка при добавлении пользователя"
fi
```

Рисунок 3 – Процесс выполнения скрипта добавления пользователя

Этот пример подчёркивает практическую ценность: скрипт минимизирует риски (например, утечки пароля), делая его подходящим для корпоративных сред. Интересно, что модель добавила опцию `-m` для создания домашней директории, предугадывая типичные нужды.

3.4 Генерация скрипта для очистки системы с опциями и статистикой

Четвёртый сценарий углубляется в очистку, где детализация промпта позволяет модели проявить креативность в добавлении фич, таких как `dry-run` и статистика. Это делает процесс увлекательным, показывая, как ИИ превращает рутину в инструмент с "пользовательским интерфейсом".

Промпт:

«Создай PowerShell-скрипт для удаления файлов старше 30 дней в `C:\Temp`. Добавь `dry-run` опцию, подсчёт удалённых файлов и логирование ошибок.»

Результат — скрипт с параметрами (`switch` для `dry-run`), рекурсивным поиском и счётчиком. Модель использовала `Remove-Item` с `-WhatIf` для симуляции, добавив `try-catch` для ошибок (Рис.4).

```
param([switch]$DryRun)

$path = "C:\Temp"
$daysOld = 30
$cutoff = (Get-Date).AddDays(-$daysOld)
$count = 0

Get-ChildItem -Path $path -Recurse -File | Where-Object { $_.LastWriteTime -lt $cutoff } | ForEach-Object {
    try {
        if ($DryRun) {
            Write-Output "Симуляция удаления: $($_.FullName)"
        } else {
            Remove-Item $_.FullName -Force
            Write-Output "Удалён: $($_.FullName)"
        }
        $count++
    } catch {
        Add-Content -Path "C:\Logs\CleanupErrors.log" -Value "$(Get-Date): Ошибка удаления $($_.FullName): $_"
    }
}

Write-Output "Обработано файлов: $count"
```

Рисунок 4 – Скрипт очистки системы

Скрипт интересен своей гибкостью: `dry-run` позволяет тестировать без риска, а статистика помогает в отчётности. Идеален для автоматизации в CI/CD пайплайнах.

3.5 Генерация скрипта для проверки служб с уведомлениями и расписанием

Пятый запрос имитирует комплексную задачу с интеграцией уведомлений, что добавляет "реального мира" — модель генерирует код, готовый к `cron` или `scheduler`. Это особенно увлекательно, поскольку показывает переход от простого скрипта к полноценному инструменту мониторинга.

Промпт:

«Напиши `bash`-скрипт для проверки статуса службы `nginx`. Если не запущена — запусти, запиши в лог и отправь email. Добавь проверку на ОС.»

GPT-5.2 создала кросс-платформенный скрипт с `systemctl` (для `systemd`), `mail` для уведомлений и `fallback` для других ОС. Добавила `timestamp` и `exit`-коды (Рис.5).

```
service="nginx"
logfile="/var/log/service_check.log"
email="admin@example.com"

if command -v systemctl >/dev/null; then
    status=$(systemctl is-active $service)
else
    status=$(service $service status >/dev/null 2>&1 && echo "active" || echo "inactive")
fi

if [ "$status" != "active" ]; then
    echo "$(date): Служба $service не запущена. Пытаюсь запустить..." >> $logfile
    if command -v systemctl >/dev/null; then
        systemctl start $service
    else
        service $service start
    fi
    if [ $? -eq 0 ]; then
        echo "$(date): $service запущена" >> $logfile
        echo "Служба $service была запущена" | mail -s "Alert: $service restarted" $email
    else
        echo "$(date): Ошибка запуска $service" >> $logfile
        echo "Ошибка запуска $service" | mail -s "Critical: $service failed" $email
    fi
else
    echo "$(date): $service в норме" >> $logfile
fi
```

Рисунок 5 – Скрипт проверки состояния служб и процессов

Этот скрипт выделяется универсальностью: работает на Linux-дистрибутивах, интегрируется с cron для периодических проверок. Полезен для поддержания uptime в веб-серверах, подчёркивая, как ИИ упрощает DevOps.

3.6 Сравнение результатов пяти сценариев

Анализ пяти примеров раскрывает закономерность: с ростом детализации промпта скрипты эволюционируют от базовых утилит к многофункциональным инструментам с безопасностью, логами и опциями. GPT-5.2 часто "улучшает" запросы, добавляя фичи вроде циклов или уведомлений, что делает её ценным помощником.

4 Выводы

Проведённое исследование убедительно показывает, что мультимодальная модель GPT-5.2 обладает высокой эффективностью при автоматической генерации скриптов для решения типичных задач системного администрирования. Модель успешно воспринимает и интерпретирует текстовые описания задач самой разной степени детализации — от самых кратких и общих формулировок до развёрнутых технических заданий — и на их основе создаёт работоспособный, хорошо структурированный код на

языках `bash` и `PowerShell`, нередко дополняя его полезными элементами, которые не были прямо указаны в запросе.

С ростом детализации исходного промпта качество генерируемых скриптов заметно повышается. Если при минимальном описании получаются простые, но вполне рабочие утилиты, то при подробном техническом задании модель выдаёт полноценные инструменты, готовые к использованию в производственной среде. В таких случаях в код автоматически включаются механизмы обработки ошибок, проверки прав доступа, логирование событий, параметры запуска, режимы сухого прогона, уведомления по электронной почте, подсчёт статистики и другие улучшения, которые делают скрипт более безопасным, удобным и надёжным. При этом модель часто проявляет проактивность: она предугадывает типичные потребности администратора, добавляя циклы для периодического мониторинга, кросс-платформенные проверки или безопасные способы ввода конфиденциальных данных.

Библиографический список

1. Суворов А. О., Суворова В. А. Интеллектуальный анализ сетевого трафика для идентификации компьютерных вторжений // Искусственный интеллект и принятие решений. 2019. № 1. С. 62–73.
2. Митрахович А. М., Митрахович Ю. М. Сравнительный анализ алгоритмов искусственного интеллекта для защиты компьютерных сетей // Хроники цифровых трансформаций : Материалы межкафедрального круглого стола, Москва, 19–26 мая 2022 года. Т. 4. Волгоград: ИП Черняева Юлия Игоревна (Издательский дом «Сириус»), 2022. С. 93–99.
3. Саенко И. Б. Искусственный интеллект: история и современное состояние исследований // Применение искусственного интеллекта в информационно-телекоммуникационных системах: сборник материалов научно-практической конференции, Санкт-Петербург, 31 марта 2021 года. СПб.: Военная академия связи имени Маршала Советского Союза С. М. Будённого, 2021. С. 5–10.
4. Долматов Е. А. Подходы к применению технологий искусственного интеллекта для решения задач автоматизированного управления в сложных организационно-технических системах // Применение искусственного интеллекта в информационно-телекоммуникационных системах: сборник материалов научно-практической конференции, Санкт-Петербург, 31 марта 2021 года. СПб.: Военная академия связи имени Маршала Советского Союза С. М. Будённого, 2021. С. 62–68.
5. Аллакин В. В., Будко Н. П., Васильев Н. В. Общий подход к построению перспективных систем мониторинга распределённых информационно-телекоммуникационных сетей // Системы управления, связи и безопасности. 2021. № 4. С. 125–227.