Headroom + RTK: как срезать ~50% расходов на токены в Claude Code

Недельный лимит Claude Code сгорает уже к среде, хотя добрая половина токенов уходит не на мысли модели, а на шум: сырые логи, JSON, вывод команд и повторяющийся контекст. Апгрейд тарифа эту проблему не решает — тот же раздутый промпт просто упирается в потолок побольше. Ниже — связка из двух локальных утилит, которая режет этот шум прямо перед тем, как он попадёт в контекст.

Важно: у Headroom CLI ставится только через PyPI. Пакет `headroom-ai` в npm — это TypeScript-библиотека для импорта в код, а не CLI, и команды `headroom` она не даёт. Перепутать легко — но это разные способы использования одного проекта.

1. Откуда берётся раздутый расход токенов

- Контекстное окно съедают не рассуждения модели, а шум: сборочные логи, JSON-блобы, вывод шелл-команд и повторяющийся boilerplate в каждом промпте.
- Повышение тарифа не лечит причину — те же раздутые промпты просто упираются в потолок побольше, и лимит всё равно кончается раньше дедлайна.
- Один запуск тестов, линтера или `grep` по проекту может закинуть в контекст тысячи «сырых» токенов там, где модели реально нужно несколько строк сути.

2. Headroom: сжатие контекста на уровне прокси

- Работает локальным прокси: перехватывает промпт перед тем, как он уйдёт в Claude Code или Codex, и обратимо сжимает логи, JSON и повторяющийся контекст — всё, что модели действительно нужно, можно вытащить обратно по запросу (механизм CCR).
- **Цифры:** по бенчмаркам на реальных сценариях экономия от 47% (обзор кодовой базы) до 92% (поиск по коду, разбор SRE-инцидента); в среднем на сессию — около 50% и вдвое больше «полезного» использования того же тарифа.
- **Установка (CLI):** `pip install "headroom-ai[all]"`, затем `headroom wrap claude` — оборачивает Claude Code прокси-слоем одной командой (github.com/headroomlabs-ai/headroom).
- **Готовое приложение:** тот же движок доступен как menu-bar приложение для macOS с 72-часовым бесплатным пробным периодом без регистрации — скачать можно на extraheadroom.com, если не хочется настраивать CLI руками.
- Все прогоны и оптимизация выполняются локально — код и промпты не покидают машину.

3. RTK: дожимает повторяющийся вывод команд

- Отдельный CLI-прокси, который фильтрует и сжимает вывод конкретных консольных команд — `git status/diff/log`, тестовые раннеры, линтеры, `docker` / `kubectl` — прежде чем он попадёт в контекст модели: умная фильтрация, группировка, обрезка и дедупликация повторов.
- **Цифры:** 60–90% экономии на частых dev-командах, например `git push` — минус 92%, тестовые раннеры вроде `cargo test` или `pytest` — минус 90%.
- **Установка:** `brew install rtk` или `curl -fsSL https://raw.githubusercontent.com/rtk-ai/rtk/refs/heads/master/install.sh | sh` (github.com/rtk-ai/rtk).
- **Подключение к Claude Code:** `rtk init -g` ставит хук, который на лету переписывает bash-команды в rtk-эквиваленты; после установки Claude Code нужно перезапустить.
- **Ограничение:** хук перехватывает только вызовы Bash — встроенные инструменты вроде Read/Grep/Glob через него не проходят, для них нужно явно вызывать `rtk read`, `rtk grep` или `rtk find`.

4. Как это работает вместе

- Headroom сжимает крупный «шумный» контекст — веб-страницы, JSON, логи, накопленную историю диалога; RTK специально дожимает частый консольный вывод git/тестов/линтеров — вместе они закрывают разные источники раздувания промпта.
- Обе утилиты работают локально и обратимо: ничего не теряется безвозвратно, полные данные всегда можно достать по запросу или из сохранённого лога.
- **Проверка результата:** `headroom doctor` и `headroom dashboard` показывают текущую экономию по Headroom; `rtk gain` — статистику по RTK.