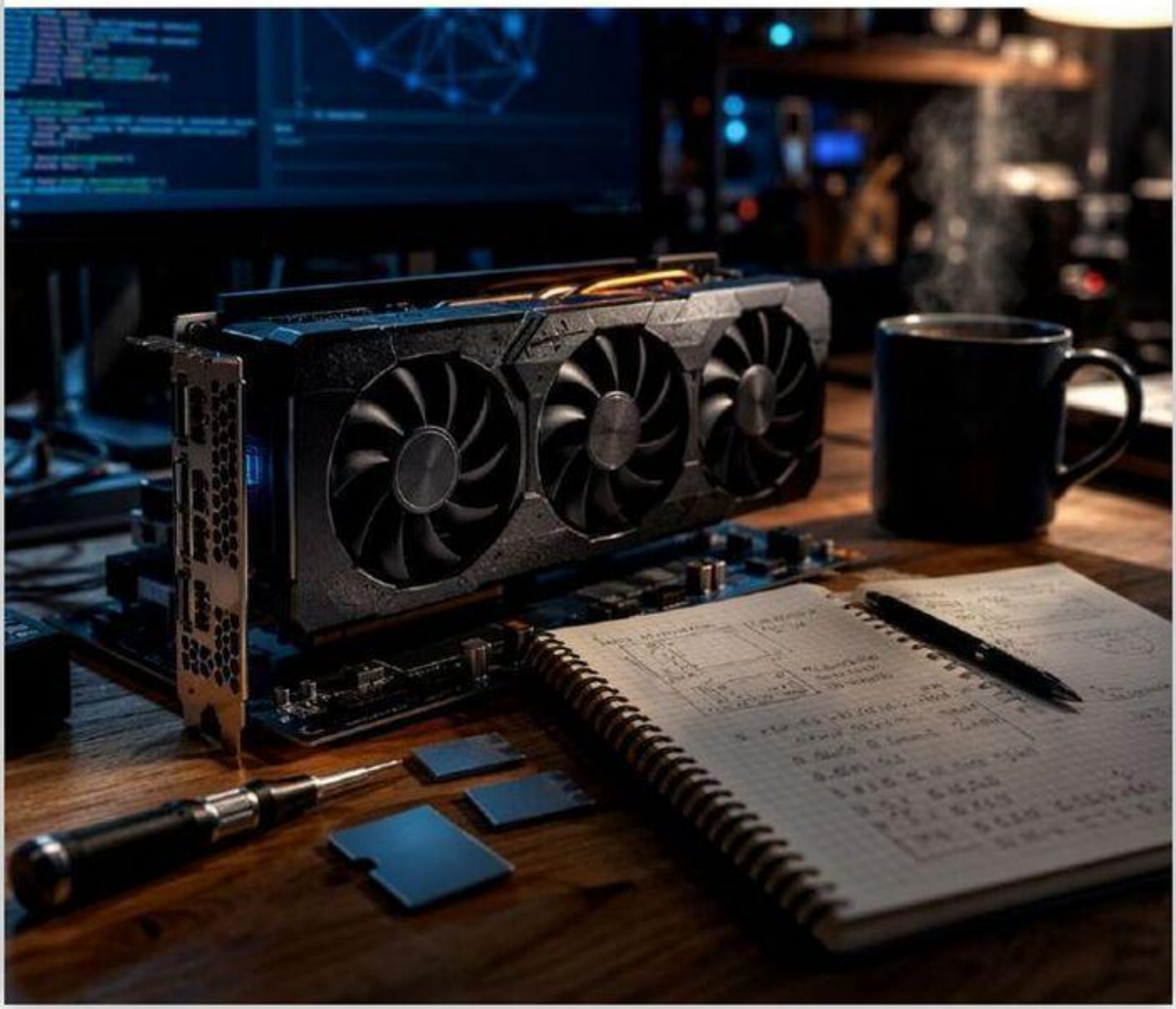
12+

Андрей Ладыгин

Локальные LLM на потребительской видеокарте

От квантизации до агентов: когда достаточно 16 ГБ VRAM?



Андрей Ладыгин

Локальные LLM на потребительской видеокарте. От квантизации до агентов: когда достаточно 16 ГБ VRAM?

«Издательские решения»

Ладыгин А.

Локальные LLM на потребительской видеокарте. От квантизации до агентов: когда достаточно 16 ГБ VRAM? / А. Ладыгин — «Издательские решения»,

Свой ИИ-стек на игровой видеокарте — без облака и апгрейда. Локальные LLM на 16 ГБ VRAM: квантизация, бюджет памяти, рантаймы, кодинг-агенты, RAG, OpenCode с tool calling. Воспроизводимые бенчмарки, диагностика, критерии апгрейда. 12 глав с упражнениями и конфигами.

© Ладыгин А.
© Издательские решения

Содержание

ЧАСТЬ I. ФУНДАМЕНТ: ПОЧЕМУ 16 ГБ — ЭТО ДОСТАТОЧНО 6
ГЛАВА 1. Анатомия VRAM: как модель, контекст и квантизация делят память 6
ГЛАВА 2. Ландшафт рантаймов: Ollama, LM Studio, llama. cpp 12
Конец ознакомительного фрагмента. 15

Локальные LLM на потребительской видеокарте От квантизации до агентов: когда достаточно 16 ГБ VRAM?

Андрей Ладыгин

Иллюстратор ИИ GPT Image 2

© Андрей Ладыгин, 2026
© ИИ GPT Image 2, иллюстрации, 2026

ISBN 978-5-0071-7049-9
Создано в интеллектуальной издательской системе Ridero

ЧАСТЬ I. ФУНДАМЕНТ: ПОЧЕМУ 16 ГБ — ЭТО ДОСТАТОЧНО

ГЛАВА 1. Анатомия VRAM: как модель, контекст и квантизация делят память

Вы скачали модель на 14 миллиардов параметров в квантизации Q5. По калькулятору на сайте модели она занимает «около 10 ГБ». У вас 16 ГБ VRAM. Вы запускаете её с контекстом 32 000 токенов — и получаете либо ошибку нехватки памяти, либо скорость 3 ток/с, потому что часть модели уехала в системную RAM. Модель «влезла», но бюджет VRAM не сошёлся.

Эта глава — про то, как считать этот бюджет. Не гадать, а считать: сколько памяти уйдёт на веса, сколько на контекст, сколько на рабочий буфер и сколько нужно оставить свободным. К концу главы вы сможете за минуту оценить, поместится ли конкретная модель с конкретным контекстом в ваши 16 ГБ.

1.1. Из чего состоит бюджет VRAM

Бюджет видеопамяти — это не одно число, а четыре статьи расхода, которые складываются. Если хотя бы одну забыть, расчёт рассыпается.

Веса модели (weights). Это то, что вы скачали. Размер зависит от числа параметров и от того, сколько бит отведено на каждый вес. Именно веса — самая крупная статья расхода, и именно их чаще всего имеют в виду, когда говорят «модель на 10 ГБ».

KV-кэш. Это память, в которой модель хранит «контекст» — ключи и значения attention для уже обработанных токенов. KV-кэш растёт линейно с длиной контекста. На коротком диалоге он почти незаметен. На 32K—128K токенах он может занять больше места, чем сама модель.

Рабочий буфер (activations и attention workspace). Промежуточные вычисления, которые нужны на каждом шаге генерации. Обычно это сотни мегабайт — гигабайт, но при больших batch size или длинных последовательностях он растёт.

Накладные расходы рантайма и ОС. Драйвер видеокарты, контекст CUDA или ROCm, фрагментация памяти, служебные буферы самого рантайма. На 16 ГБ карте это обычно 0,5—1 ГБ, которые вы не увидите в калькуляторе модели.

Под капотом. На Windows часть VRAM резервируется под WDDM (Windows Display Driver Model), поэтому «доступно» всегда меньше «установлено». На Linux накладные расходы обычно ниже, но зависят от драйвера и версии CUDA/ROCm.

Соберём все статьи расхода в один расчёт — и получим реальный бюджет. Правило, которое стоит запомнить с этой главы: **оставляйте около 15% VRAM свободными** (это эвристика автора, а не абсолют). Этот запас покрывает фрагментацию, пики рабочего буфера и накладные расходы, которые вы не учли.



Рис. 1.1. Бюджет VRAM: четыре статьи расхода и запас 15%

1.2. Веса: сколько на самом деле весит модель

Оценка размера весов — арифметика, которую полезно уметь делать в уме.

Формула: размер весов ≈ (число параметров × бит на вес) / 8.

Для 7B в FP16: $7 \times 10^9 \times 16 / 8 \approx 14$ ГБ. Для 7B в Q4: $7 \times 10^9 \times 4 / 8 \approx 3{,}5$ ГБ по формуле; реальный файл GGUF обычно 4,0—4,5 ГБ из-за служебных тензоров и метаданных.

Вот ориентировочная таблица для типичных квантизаций. Точные размеры зависят от реализации квантизации (Q, K-quants, IQ) и от архитектуры модели.

Квантизация | 7B | 12B | 14B | 24B | 27B | 32B
FP16 | ~14 ГБ | ~24 ГБ | ~28 ГБ | ~48 ГБ | ~54 ГБ | ~65 ГБ
Q8_0 | ~7 ГБ | ~12,8 ГБ | ~14,6 ГБ | ~23,7 ГБ | ~27 ГБ | ~34 ГБ
Q6_K | ~5,5 ГБ | ~9,8 ГБ | ~11,3 ГБ | ~18,3 ГБ | ~22 ГБ | ~26 ГБ
Q5_K_M | ~4,8 ГБ | ~8,5 ГБ | ~9,8 ГБ | ~15,9 ГБ | ~19 ГБ | ~23,1 ГБ
Q4_K_M | ~4 ГБ | ~7,3 ГБ | ~8,4 ГБ | ~13,6 ГБ | ~16 ГБ | ~19,5 ГБ
Q3_K_M | ~3,1 ГБ | ~5,8 ГБ | ~6,8 ГБ | ~10,5 ГБ | ~12 ГБ | ~15,9 ГБ
Q2_K | ~2,2 ГБ | ~4,5 ГБ | ~5,5 ГБ | ~8,5 ГБ | ~9,5 ГБ | ~12,3 ГБ

Источник. 12B — карточка Gemma 4 12B (Q4_K_M 7,3 ГБ, Q5_K_M 8,5 ГБ; canitrun.dev). 14B — карточка DeepSeek-R1-Distill-Qwen-14B (Q4_K_M 8,37 ГБ, Q5_K_M 9,79 ГБ; modelfit.io). 24B — карточка Mistral Small 3.x 24B (Q4_K_M 13,6 ГБ, Q5_K_M 15,9 ГБ, Q3_K_M 10,5 ГБ; cooconsbit.com). 32B — карточка Qwen2.5—32B (Q4_K_M 19,5 ГБ, Q5_K_M 23,1 ГБ; gigagpu.com). 7B и 27B — [НУЖЕН ИСТОЧНИК: добавить карточки для Llama-3.1—8B и Qwen3.8—27B].

Из таблицы видно главное: **24B dense в Q4_K_M — это ~13,6 ГБ только весов**. На 16 ГБ карте такая модель оставляет ~2—3 ГБ под KV-кэш и буфер. Это возможно при коротком контексте, но не при 32K.

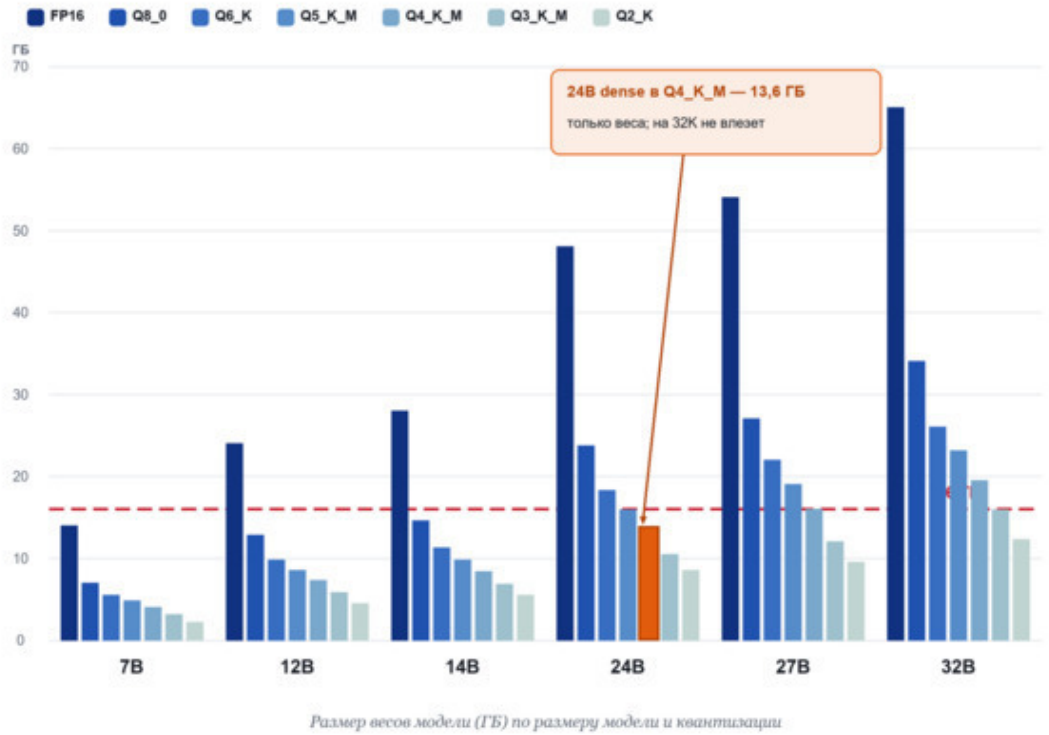


Рис. 1.2. Размер весов по модели и квантизации

Отдельно про MoE. У Mixture-of-Experts моделей есть два разных числа: полное число параметров (все эксперты) и число активных параметров (те, что реально участвуют в генерации одного токена). Для скорости важны активные параметры, **для VRAM — полное число**, потому что все эксперты должны где-то храниться. 30B MoE с 3B активных параметров в Q4 занимает ~17 ГБ — почти столько же, сколько 27B dense.

Под капотом. K-quants (Q4_K_M, Q5_K_M) — это блочное квантование с разной точностью для разных тензоров: более важные слои квантуются мягче. IQ-кванты (IQ2_XS, IQ3_XXS) используют решёточное квантование и дают лучшее качество при том же размере, но требуют поддержки со стороны рантайма.

Типичная ошибка. Смотреть на «активные параметры» MoE и думать, что модель займёт столько же, сколько dense-модель с таким числом параметров. Для VRAM решает полный размер.

1.3. KV-кэш: главный скрытый пожиратель

KV-кэш — это та часть бюджета, которую новички забывают чаще всего. Именно из-за неё «модель, которая влезала вчера» перестаёт влезать сегодня, когда вы поднимаете контекст с 8K до 32K.

Что хранится в KV-кэше: для каждого слоя attention и для каждого уже обработанного токена — ключ (key) и значение (value) по каждой KV-голове attention. Формула оценки:

KV-кэш ≈ 2 × слои × KV-головы × head_dim × токены × байт_на_элемент

Множитель 2 — потому что хранятся и ключи, и значения. байт_на_элемент зависит от точности KV-кэша: FP16 — 2 байта, Q8 — 1 байт, Q4 — 0,5 байта.

Возьмём Qwen3 8B: 36 слоёв, 8 KV-голов (GQA), head_dim 128. Для 8K токенов в FP16:

2 × 36 × 8 × 128 × 8192 × 2 ≈ 1,2 ГБ

Для 32K — уже ~4,8 ГБ. Для 128K — ~19 ГБ, что само по себе больше 16 ГБ.

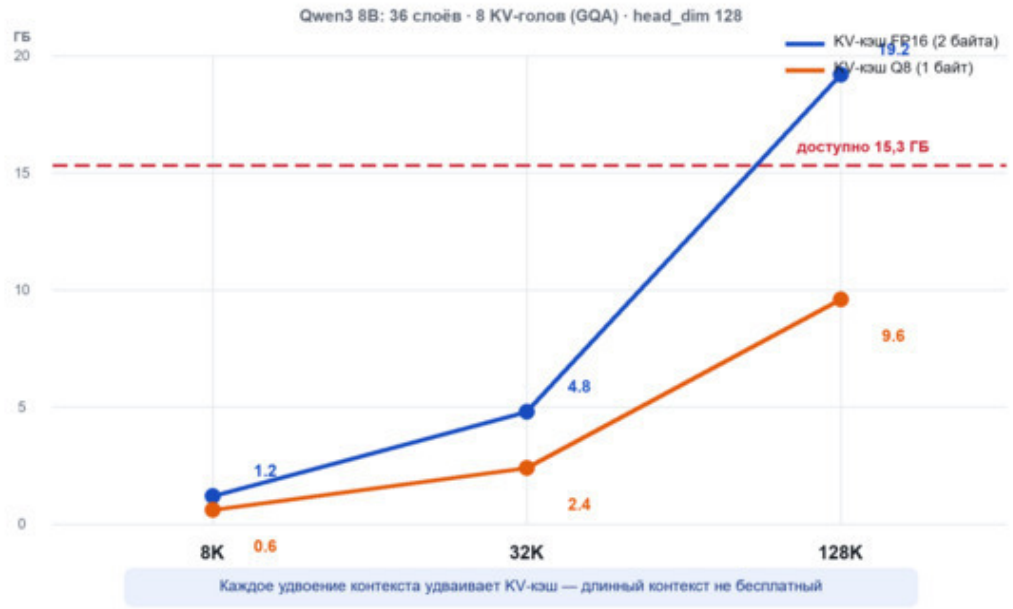


Рис. 1.3. Рост KV-кэша с длиной контекста (Qwen3 8B: FP16 и Q8)

Под капотом. GQA (Grouped-Query Attention) и MQA (Multi-Query Attention) сокращают число KV-голов, для которых хранится KV-кэш, во много раз. Например, Qwen3 8B с 8 KV-головами в bf16 даёт 144 KiB KV-кэша на токен. MHA-версия с 32 KV-головами потребовала бы в 4 раза больше.

Отсюда практическое правило: **длинный контекст — это не бесплатно**. Каждое удвоение контекста удваивает KV-кэш. И квантизация KV-кэша (например, Q8) снижает его вдвое, но может стоить небольшой потери качества.

Типичная ошибка. Ставить контекст 128K «на всякий случай», потому что модель его поддерживает. Поддержка — не то же самое, что помещается в вашу карту вместе с весами.

1.4. Рабочий буфер и накладные расходы

Две статьи, которые легко игнорировать, но которые складываются в гигабайты.

Рабочий буфер (activations). Промежуточные тензоры, которые создаются на каждом шаге прямого прохода. Их размер зависит от batch size и длины последовательности. При batch size 1 и коротком контексте это сотни мегабайт. При больших batch или длинном prefill (обработке входного промпта перед генерацией) — больше.

Attention workspace. Дополнительная память под вычисление attention. Flash-attention существенно снижает её, но не обнуляет. На длинных последовательностях и без flash-attention workspace может достигать нескольких гигабайт.

Накладные расходы. Драйвер, контекст CUDA/ROCm, служебные буферы рантайма, фрагментация. На практике из 16 ГБ «установлено» вам доступно ~15,3 ГБ (ориентировочно; зависит от ОС и драйвера). [ПРОВЕРИТЬ: точные значения на разных ОС.]

Проверь себя. Запустите nvidia-smi (или rocm-smi) до загрузки модели и после. Разница между «занято» и размером весов — это ваш рабочий буфер плюс накладные расходы. Запишите это число для своей системы. Ожидаемый ответ: 0,5—1,5 ГБ в зависимости от ОС и рантайма.

1.5. Собираем бюджет: практический калькулятор

Теперь сложим всё вместе. Алгоритм из пяти шагов:

1. **Веса.** Возьмите число параметров и квантизацию, оцените размер по формуле (или по таблице выше).
2. **KV-кэш.** Оцените по формуле для вашей длины контекста и точности KV-кэша.
3. **Рабочий буфер.** Добавьте 0,5—1,5 ГБ в зависимости от длины prefill и batch.
4. **Накладные расходы.** Добавьте 0,5—1 ГБ на драйвер и рантайм.
5. **Запас.** Убедитесь, что сумма ≤ 85% доступной VRAM.

Вот как это выглядит для трёх типичных сценариев на 16 ГБ (доступно ~15,3 ГБ, рабочий запас — 15%):

Сценарий | Контекст | Модель | Веса | KV (FP16) | Буфер + накладные | Итог | Вердикт

Чат | 8K | Gemma 4 12B Q4_K_M | ~7,3 ГБ | ~0,5 ГБ | ~1,5 ГБ | ~9,3 ГБ | Влезает

RAG | 32K | DeepSeek-R1—14B Q4_K_M | ~8,4 ГБ | ~3,0 ГБ | ~2 ГБ | ~13,4 ГБ | Влезает

RAG | 32K | Mistral Small 24B Q4_K_M | ~13,6 ГБ | ~5,0 ГБ | ~2 ГБ | ~20,6 ГБ | Не влезает

RAG | 32K | Mistral Small 24B Q3_K_M | ~10,5 ГБ | ~5,0 ГБ | ~2 ГБ | ~17,5 ГБ | На грани

Агент | 64K | gpt-oss 20B MXFP4 | ~12,1 ГБ | ~2,0 ГБ | ~2 ГБ | ~16,1 ГБ | На грани; нужен Q8 KV

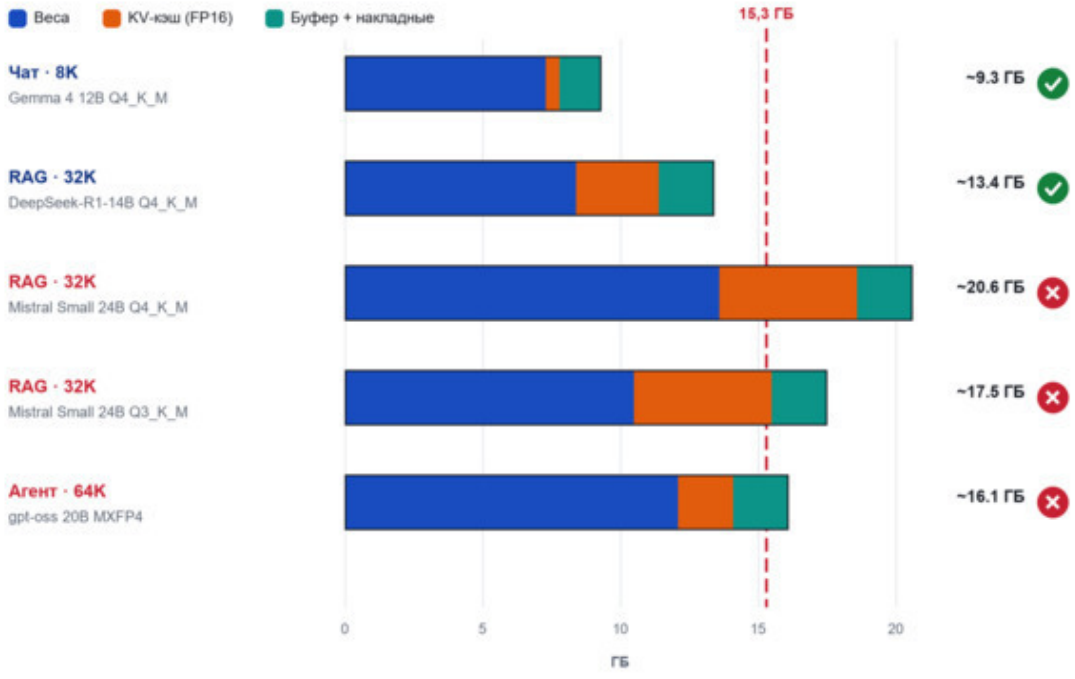


Рис. 1.4. Бюджет VRAM для типичных сценариев на 16 ГБ

Источник. Размеры весов подтверждены карточками моделей (см. раздел 1.2). KV-кэш для Mistral 24B: 16K ~2,5 ГБ, 32K ~5,0 ГБ (cooconsbit.com). gpt-oss 20B MXFP4: 12,1 ГБ (GitHub).

Типичная ошибка. Считать только веса и удивляться, почему 32K контекст «не работает». KV-кэш на 32K для 24B-модели — это ~5 ГБ, которые нужно вычесть из бюджета до того, как вы добавите буфер и накладные.

Итоги главы

1. Бюджет VRAM = веса + KV-кэш + рабочий буфер + накладные расходы.
2. Квантизация весов — главный рычаг экономии, но не единственный.
3. KV-кэш растёт линейно с контекстом; на длинном контексте он может занять больше места, чем сама модель.
4. MoE-модели для VRAM считаются по полному числу параметров, а не по активным.
5. Оставляйте ~15% VRAM свободными (эвристика автора): запас покрывает фрагментацию и пики буфера.

Упражнения

1. Оцените размер весов для 14B в Q6_K и Q4_K_M. Сравните с таблицей.
2. Для Mistral Small 24B посчитайте KV-кэш на 8K, 32K и 128K в FP16 и Q8.
3. Запустите nvidia-smi/rocm-smi до и после загрузки модели. Запишите накладные расходы вашей системы.
4. Возьмите сценарий «RAG на 32K» и подберите комбинацию модели и квантизации, которая влезает в ваши 16 ГБ с запасом 15%.

Что дальше

Теперь, когда мы умеем считать бюджет VRAM, следующий вопрос — какой рантайм даёт лучший контроль над этим бюджетом. В Главе 2 разберём Ollama, LM Studio и llama.cpp: чем они отличаются, какие флаги управляют выгрузкой слоёв (offload) и KV-кэшем, и для каких сценариев каждый из них подходит.

ГЛАВА 2. Ландшафт рантаймов: Ollama, LM Studio, llama. cpp

Три пользователя запускают одну и ту же модель на одной и той же карте с 16 ГБ VRAM. Первый получает 45 ток/с и стабильную работу. Второй — 30 ток/с и периодические OOM. Третий — 12 ток/с и выгрузку половины модели в RAM. Модель одна. Карта одна. Разница — в рантайме.

Эта глава — про три основных рантайма для локальных LLM: Ollama, LM Studio и llama. cpp. Мы разберём, что у них общего, чем они отличаются, и как выбрать тот, который соответствует вашей задаче.

2.1. Что общего у всех рантаймов

Все три рантайма работают с одним форматом файлов — GGUF (GPT-Generated Unified Format). Это контейнер, в котором лежат веса модели в квантизованном виде плюс метаданные: архитектура, число слоёв, параметры токенизатора. Если вы скачали модель с Hugging Face в формате GGUF, она подойдёт любому из трёх рантаймов.

Все три умеют выгружать слои в системную RAM, когда VRAM не хватает. В llama. cpp это флаг — n-gpu-layers, в Ollama — параметр num_gpu в Modelfile, в LM Studio — ползунок GPU offload. Механизм один: часть слоёв остаётся на GPU, часть уезжает в RAM, и скорость падает, потому что пропускная способность RAM ниже, чем у VRAM.

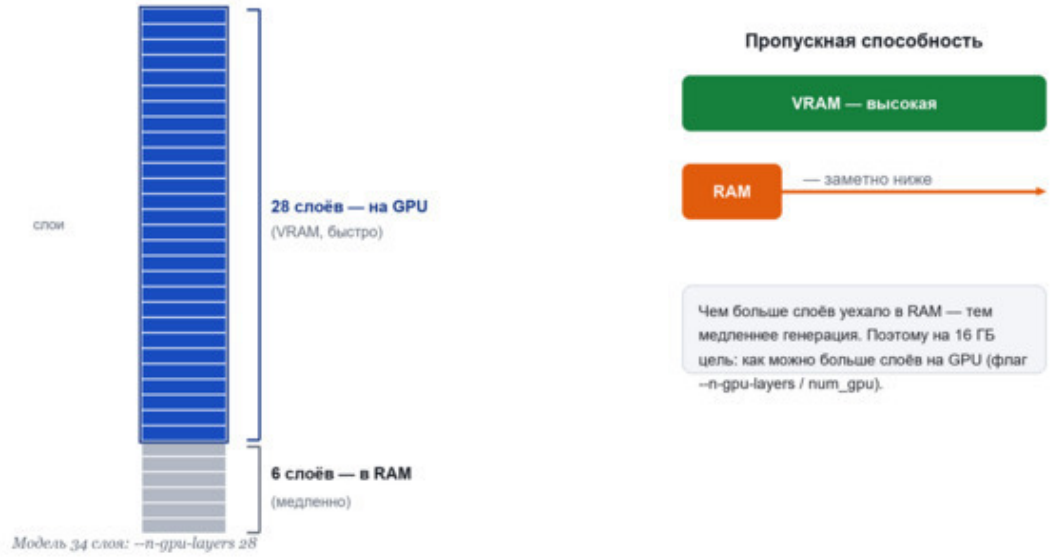


Рис. 2.1. Offload: слои на GPU и в RAM

Все три могут поднять OpenAI-совместимый endpoint. Это значит, что любой инструмент, который умеет говорить с OpenAI API (включая OpenCode), может работать с локальной моделью через этот endpoint.

Различия — в контроле, удобстве и экосистеме. Именно им посвящены следующие три раздела.



Рис. 2.2. Три рантайма и один OpenAI-совместимый протокол

2.2. Ollama: агентский стандарт

Ollama — это рантайм, который чаще всего выбирают для агентских сценариев. Причина — в балансе простоты и функциональности.

Modelfile. Конфигурация модели в Ollama описывается декларативно. Вы создаёте текстовый файл, в котором указываете базовую модель, параметры запуска и системный промпт. Вот пример для 14B-модели в Q4_K_M:

```
FROM deepseek-r1:14b-q4_K_M
PARAMETER num_gpu 28
PARAMETER num_ctx 16384
PARAMETER temperature 0.7
SYSTEM «Ты — технический ассистент. Отвечай кратко и по делу.»
```

num_gpu 28 означает, что 28 слоёв останутся на GPU, остальные уедут в RAM. num_ctx 16384 задаёт длину контекста. Эти два параметра — главные рычаги управления бюджетом VRAM в Ollama.

Встроенный сервер. Ollama поднимает OpenAI-совместимый endpoint на localhost:11434. Это значит, что OpenCode, Continue, Cline и другие агентские инструменты могут подключиться к нему без дополнительной настройки.

Сильные стороны. Простота установки, автоуправление памятью, готовая интеграция с агентскими инструментами.

Ограничения. Меньше контроля над KV-кэшем и flash-attention, чем в llama. cpp. Некоторые продвинутые флаги недоступны.

Под капотом. Ollama автоматически определяет, сколько слоёв можно загрузить на GPU, исходя из доступной VRAM. Это удобно, но иногда консервативно: она может оставить на GPU меньше слоёв, чем поместилось бы.

2.3. LM Studio: GUI для экспериментов

LM Studio — это рантайм с графическим интерфейсом. Он удобен для быстрого старта и экспериментов, но менее удобен для автоматизации.

GUI. Вы загружаете модель через интерфейс, настраиваете длину контекста, ползунком выбираете число слоёв на GPU, видите текущее потребление VRAM в реальном времени.

Встроенный сервер. LM Studio поднимает OpenAI-совместимый endpoint на localhost:1234.

Сильные стороны. Наглядность, быстрый старт, удобство для экспериментов с разными моделями и квантизациями.

Ограничения. Меньше автоматизации, не для продакшн-пайплайнов. Настройки хранятся в GUI, а не в конфигурационных файлах.

2.4. llama. cpp: полный контроль

llama. cpp — это низкоуровневый рантайм, который даёт максимальный контроль над всеми параметрами. Он сложнее в настройке, но именно он позволяет получить максимальную производительность из 16 ГБ VRAM.

llama-server. Это компонент llama. cpp, который поднимает OpenAI-совместимый endpoint на localhost:8080. Вот пример запуска для 14B-модели с контекстом 32K и квантизованным KV-кэшем:

```
llama-server \
— m deepseek-r1-14b-q4_K_M.gguf \
— n-gpu-layers 28 \
— ctx-size 32768 \
— flash-attn \
— cache-type-k q8_0 \
— cache-type-v q8_0 \
— host 0.0.0.0 \
— port 8080
```

Флаг — n-gpu-layers 28 оставляет 28 слоёв на GPU. — ctx-size 32768 задаёт контекст. — flash-attn включает flash-attention, который снижает потребление памяти под attention workspace. — cache-type-k q8_0 и — cache-type-v q8_0 квантизуют KV-кэш до 8 бит, что вдвое снижает его размер по сравнению с FP16.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, купив полную легальную версию на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.