Ачинский техникум
НЕФТИ и ГАЗА
имени Е.А. Демьяненко

АТНГ

Курсовая работа

09.02.07 Информационные системы и программирование

Разработка нейросетевого движка «Xenith»

Выполнил: _____

Группа: _____

Преподаватель: _____

Цель и задачи

Цель — разработать в Xenith режим компактного хранения GGUF-весов в Vulkan-памяти без полной FP32-распаковки.

Задачи:

1. проанализировать загрузку GGUF и хранение Tensor в Xenith и llama.cpp;
2. реализовать packed-хранилище и выбор weight_type (source/f32/f16/bf16/q8_0/q4_0/q4_1);
3. добавить Vulkan pinned-буферы и packed GEMM/embed-шейдеры;
4. унифицировать CLI, Python и Ollama API;
5. проверить память, CPU fallback, сохранение и границу обучения.

Описание проекта

Xenith — локальный нейросетевой движок на C++17 для обучения, инференса и агентных сценариев.

- Форматы: .xnh, GGUF v1–v3, safetensors; архитектура LLaMA/MHA/GQA.
- Бэкенды: CPU и Vulkan compute; общий Config/Params/Tensor.
- Packed-веса: source сохраняет тип GGUF, явный тип перепаковывается при загрузке.
- Нормы F32; сложные fused/MoE/обучение — FP32 fallback.
- Интерфейсы: CLI, Python xenith_engine, Ollama-compatible HTTP.

Архитектура и путь данных

01 · Загрузка и выбор формата

От интерфейса до packed-метаданных: исходный тип, точная раскладка, без полного FP32-unpack

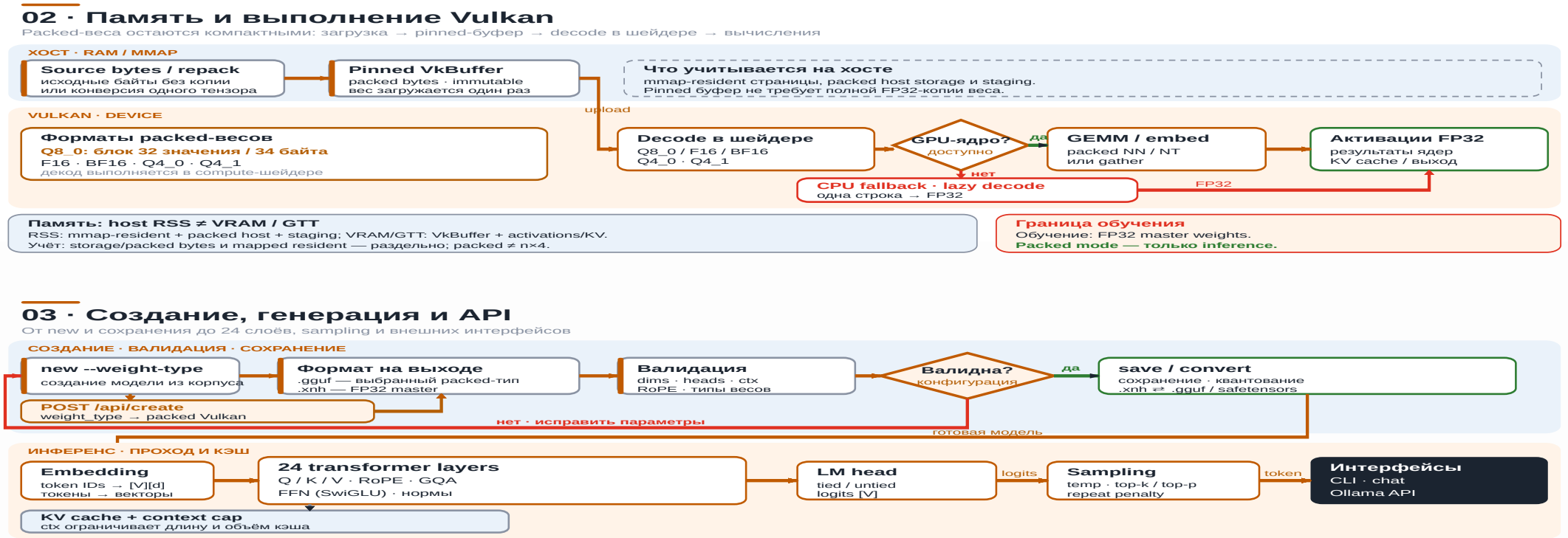


Важно

Нормы остаются F32; source сохраняет исходный тип каждого тензора.

Полного FP32-unpack нет: packed-декодирование ограничено нужной строкой или тензором.

Память Vulkan и генерация



Возможности и сравнение

Xenith ориентирован на единый цикл модель → обучение → инференс → API, llama.cpp — на зрелый высокопроизводительный GGML-инференс.

XENITH

- CPU+Vulkan, .xnh/GGUF/ST
- source/явный packed weight_type
- обучение, Python, Ollama API
- агентный/MoE слой

LLAMA.CPP

- широкий GGML backend стек
- scheduler, KV-типы, FA, MoE
- оптимизированные квант-ядра
- mature offload/multi-GPU

BASELINE

- MiniCPM5 Q8_0, ctx128,n1
- Xenith до: RSS guard 1.55GiB
- llama.cpp ngl24: RSS ~1.20GiB
- post-замер нужен отдельно

Заключение

Реализован путь GGUF → packed Tensor → Vulkan без полной FP32-копии для поддерживаемых dense-матриц.

- source сохраняет исходные типы GGUF; явные типы: f32/f16/bf16/q8_0/q4_0/q4_1.
- packed GEMM/embedding в шейдерах; pinned device buffers; lazy FP32 fallback.
- CLI/Python/Ollama используют единый weight_type; .xnh и обучение остаются FP32.
- Baseline: Xenith до изменения — guard 1.55 GiB; llama.cpp ngl24 — RSS ~1.20 GiB, GTT spill.
- **Post-реализационные замеры производительности/качества требуют отдельного воспроизводимого прогона.**