education

Оптимизация VLM

Курс “Мультимодальные модели”

Кирилл Владимирович Каймаков

Кандидат компьютерных наук
Tech Lead группы мультимодальных языковых моделей VK
Преподаватель департамента политики и управления НИУ ВШЭ

education

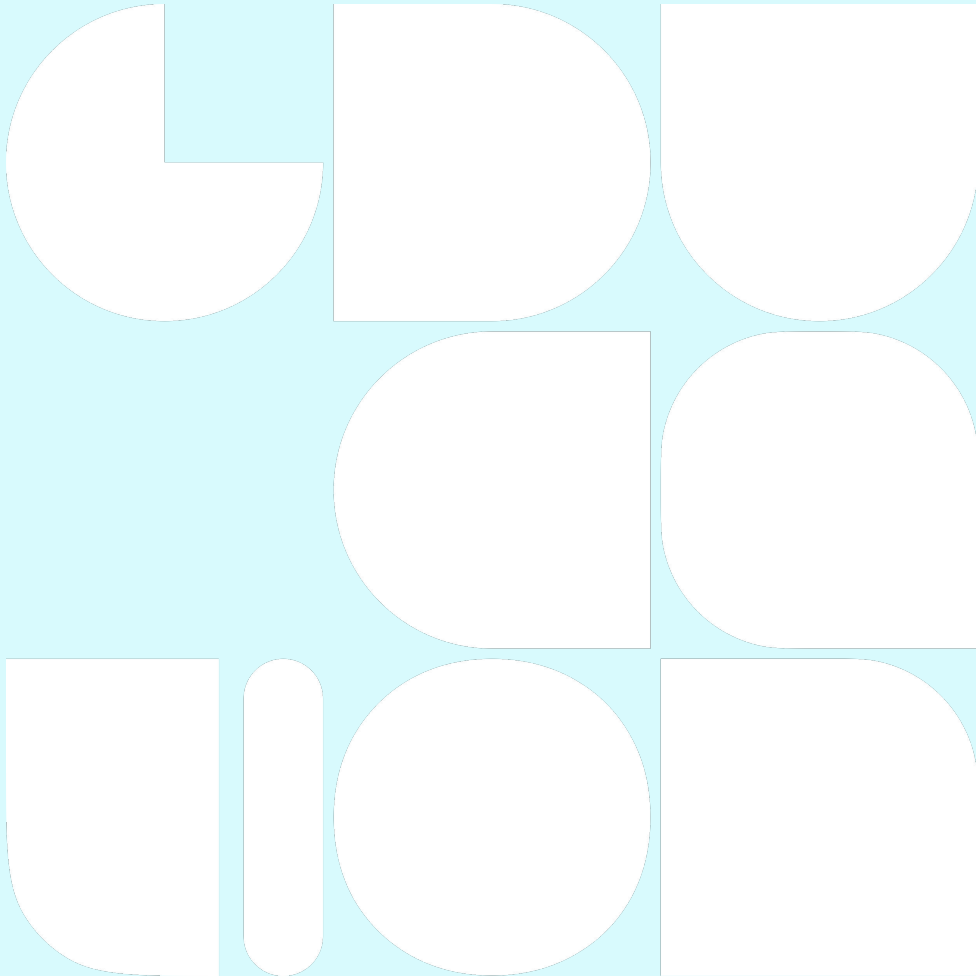
education

Что можно оптимизировать в VLM?

О чём поговорим?

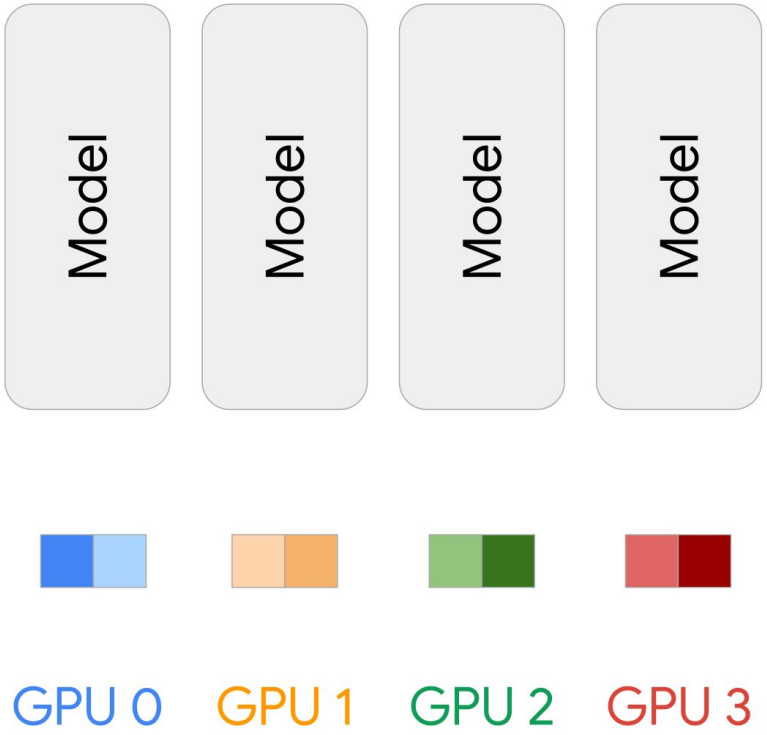
1. Параллелизация
2. Оптимизации обучения
3. Квантизации
4. Оптимизации инференса
5. Фреймворки

1. Параллелизация
2. Оптимизации обучения
3. Квантизации
4. Оптимизации инференса
5. Фреймворки



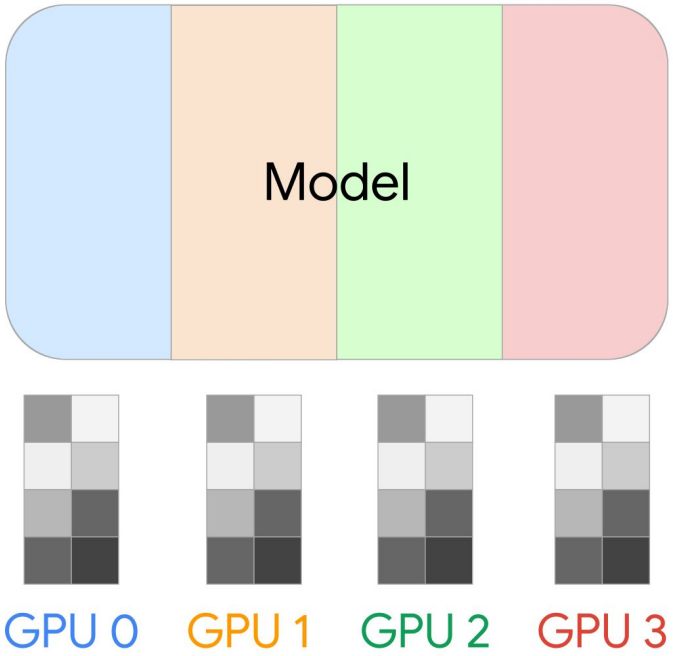
Data parallelism

1. Разделение данных: Обучающий батч разбивается на части, каждая отправляется на отдельное устройство.
2. Вычисление градиентов: Все устройства делают прямой и обратный проход по своей части данных.
3. Синхронизация: Градиенты между устройствами объединяются (обычно усредняются).
4. Обновление весов: Все копии модели получают одинаковые обновлённые параметры.



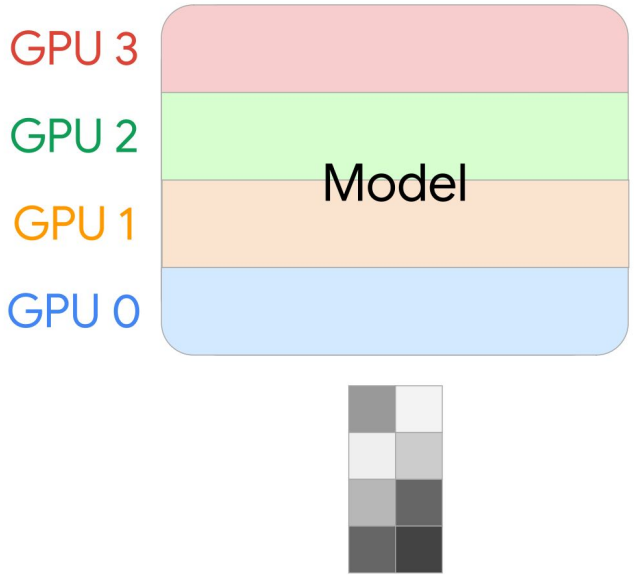
Tensor parallelism

1. Деление параметров: Тензоры весов (например, матрицы в слоях) разбиваются на части по определённой размерности и распределяются по устройствам.
2. Совместные вычисления: Прямой и обратный проходы выполняются совместно: каждое устройство вычисляет свою часть, а затем результаты собираются (например, через операции all-reduce или all-gather).
3. Обновление весов: Каждый девайс обновляет только свою часть параметров.



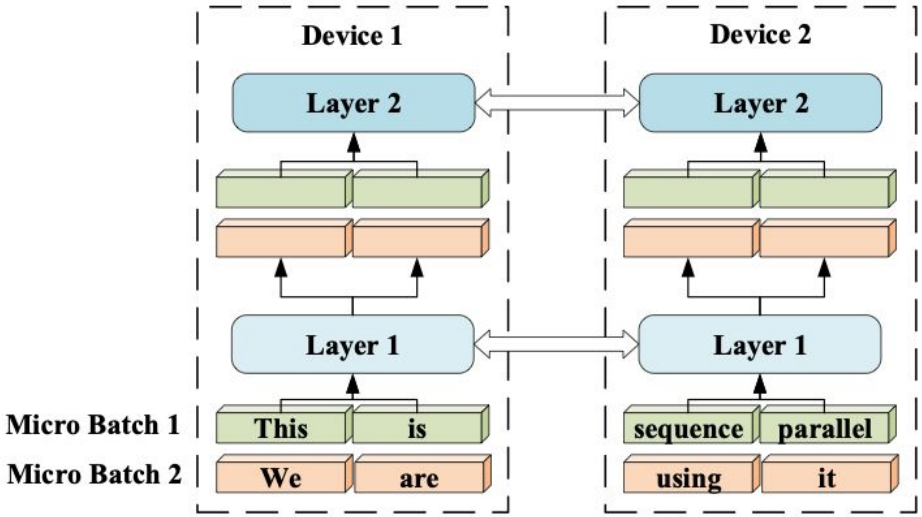
Pipeline parallelism

1. Деление модели: Модель разбивается на несколько блоков (стейджей), каждый из которых размещается на отдельном устройстве.
2. Поток данных: Мини-батч разбивается на микробатчи. Первый микробатч проходит через первый стейдж, затем передаётся на второй и так далее.
3. Параллельная обработка: Пока первый микробатч обрабатывается на втором стейдже, первый стейдж уже начинает обрабатывать следующий микробатч. Так достигается "конвейерный эффект".
4. Синхронизация: После обработки всех микробатчей происходит обратное распространение ошибки (backward pass) по тем же стейджам.



Sequence parallelism

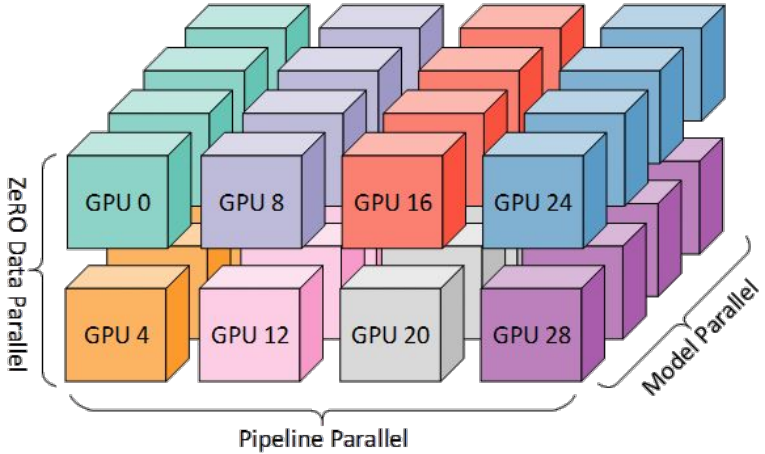
1. Деление последовательности: Входная последовательность (например, предложение из 1024 токенов) разбивается на части по токенам, и каждая часть отправляется на отдельное устройство.
2. Совместные вычисления: Каждый GPU вычисляет свою часть активаций и градиентов. Для некоторых операций (например, self-attention) требуется обмен информацией между устройствами (обычно через all-gather).
3. Сбор результатов: После вычислений части результатов объединяются для получения полного выхода слоя.



Complex parallelism

Data Parallelism + Tensor Parallelism + Pipeline Parallelism называется 3D Parallelism

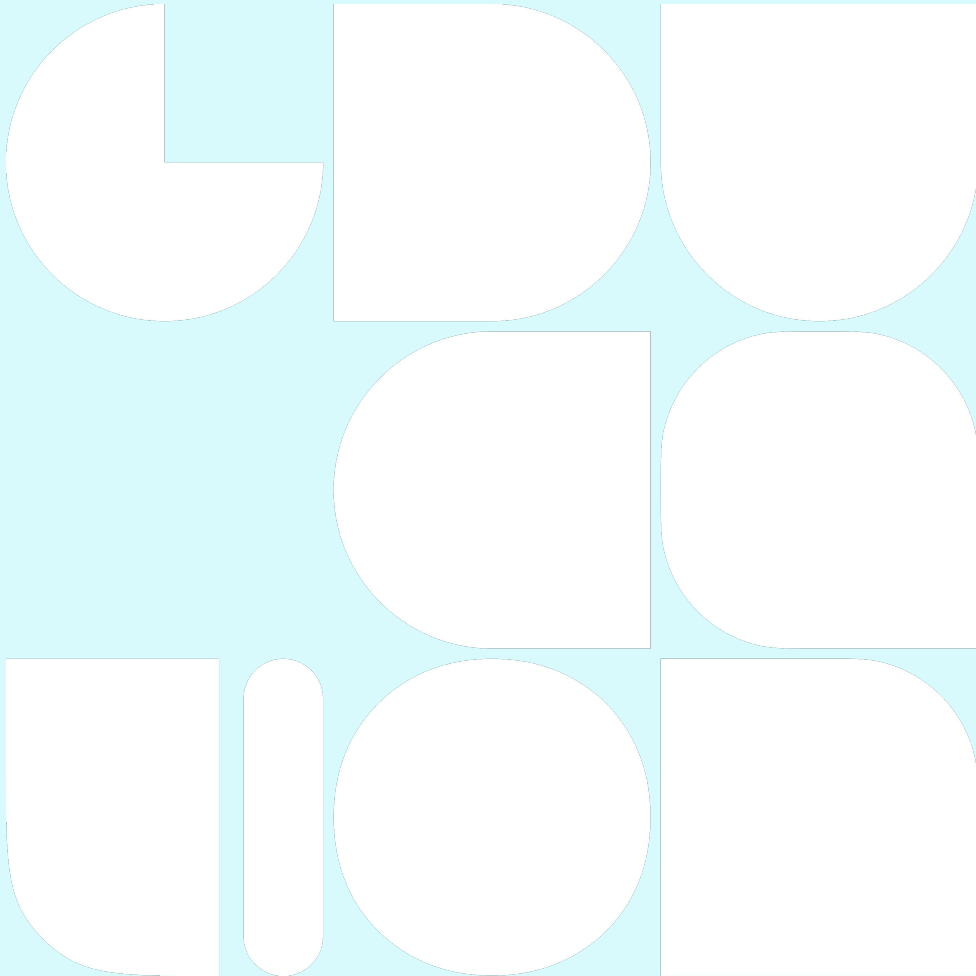
1. Модель и данные одновременно разбиваются по разным осям:
   a. Data Parallelism: копии модели обучаются на разных подвыборках данных.
   b. Tensor Parallelism: веса слоёв делятся между устройствами.
   c. Pipeline Parallelism: модель разбивается на последовательные стейджи.
   d. Sequence Parallelism: длинные последовательности делятся между устройствами.
2. Согласованная коммуникация: Все типы параллелизма требуют синхронизации и обмена информацией между устройствами на разных этапах обучения.
3. Масштабирование: Позволяет использовать тысячи GPU/TPU, эффективно распределяя как вычисления, так и память.



education

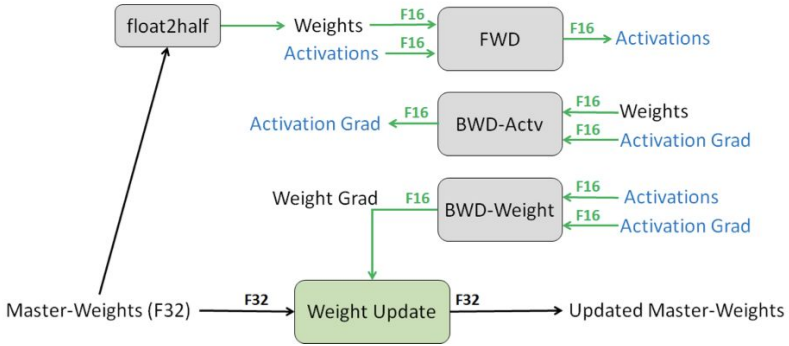
Какой из способов параллелизма работает на каком уровне суперкомпьютера?

1. Параллелизация
2. Оптимизации обучения
3. Квантизации
4. Оптимизации инференса
5. Фреймворки



Mixed precision training (2017)

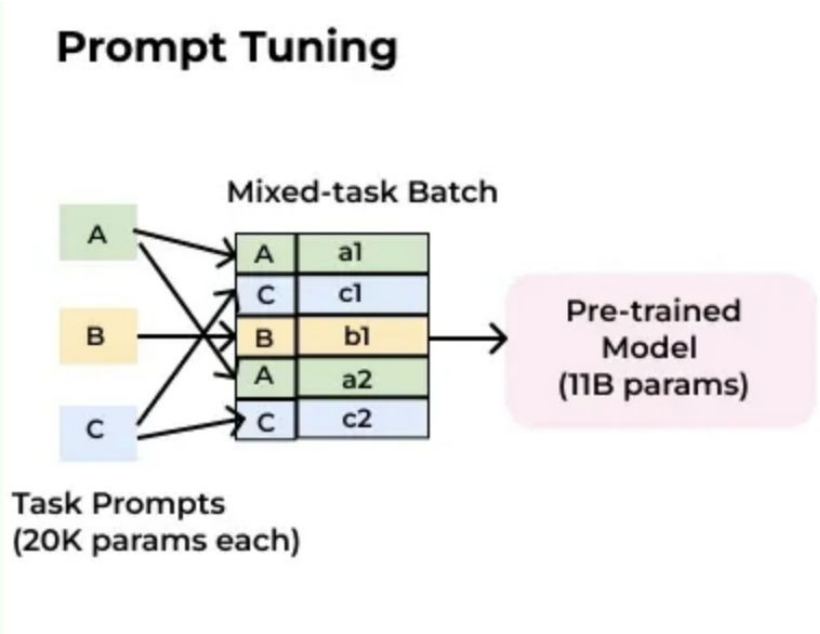
1. Большая часть вычислений (умножения матриц, свёртки и т.д.) выполняется в формате FP16 или BF 16 (16-битные числа с плавающей запятой).
2. Критически важные операции (например, накопление градиентов, обновление весов) выполняются в FP32 (32-битные числа) для сохранения точности и стабильности.
3. Используются специальные алгоритмы (например, loss scaling), чтобы избежать проблем с переполнением или исчезающими градиентами.



[Mixed Precision Training]

Prompt tuning (2021)

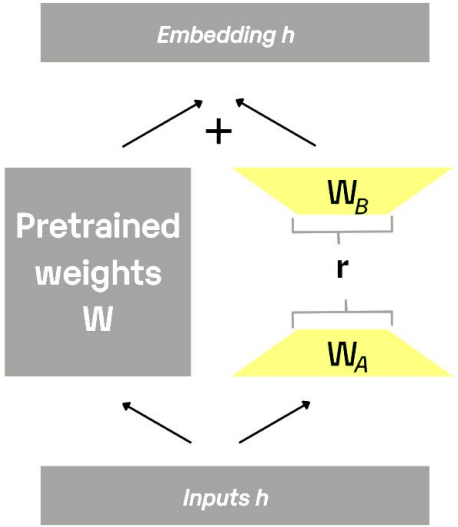
1. Вместо дообучения всех параметров модели создаётся набор обучаемых эмбеддингов.
2. Эти эмбеддинги добавляются к входной последовательности (например, в начало текста или к визуальным токенам).
3. Модель остаётся замороженной, а оптимизируются только параметры промпта.
4. Для разных задач можно обучать разные промпты, не меняя основную модель.



[The Power of Scale for Parameter-Efficient Prompt Tuning]

LoRA (2023)

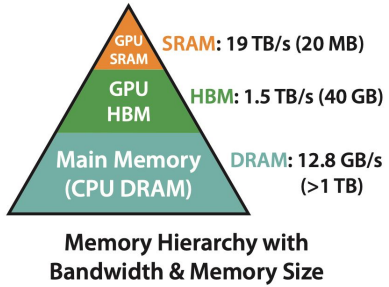
1. Основная матрица весов не изменяется.
2. В параллельную ветку добавляются две маленькие матрицы A и B, реализующие низкоранговое преобразование.
3. Итоговые веса W′ = W + A@B:
4. Обучаются только A и B, что сильно экономит память и ускоряет обучение.
5. Перед инференсом расчитывается W′



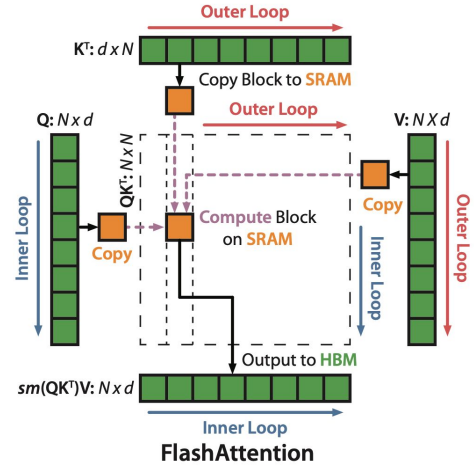
[LoRA: Low-Rank Adaptation of Large Language Models]

Flash Attention

1. Блочная обработка: Attention вычисляется по частям (блоками), чтобы не хранить всю матрицу attention в памяти.
2. Оптимизация доступа к памяти: Использует эффективные схемы чтения/записи, минимизируя обращения к медленной памяти.
3. Вычисления "на лету": Softmax и суммы считаются сразу для каждого блока, без промежуточного хранения больших тензоров.



Memory Hierarchy with Bandwidth & Memory Size



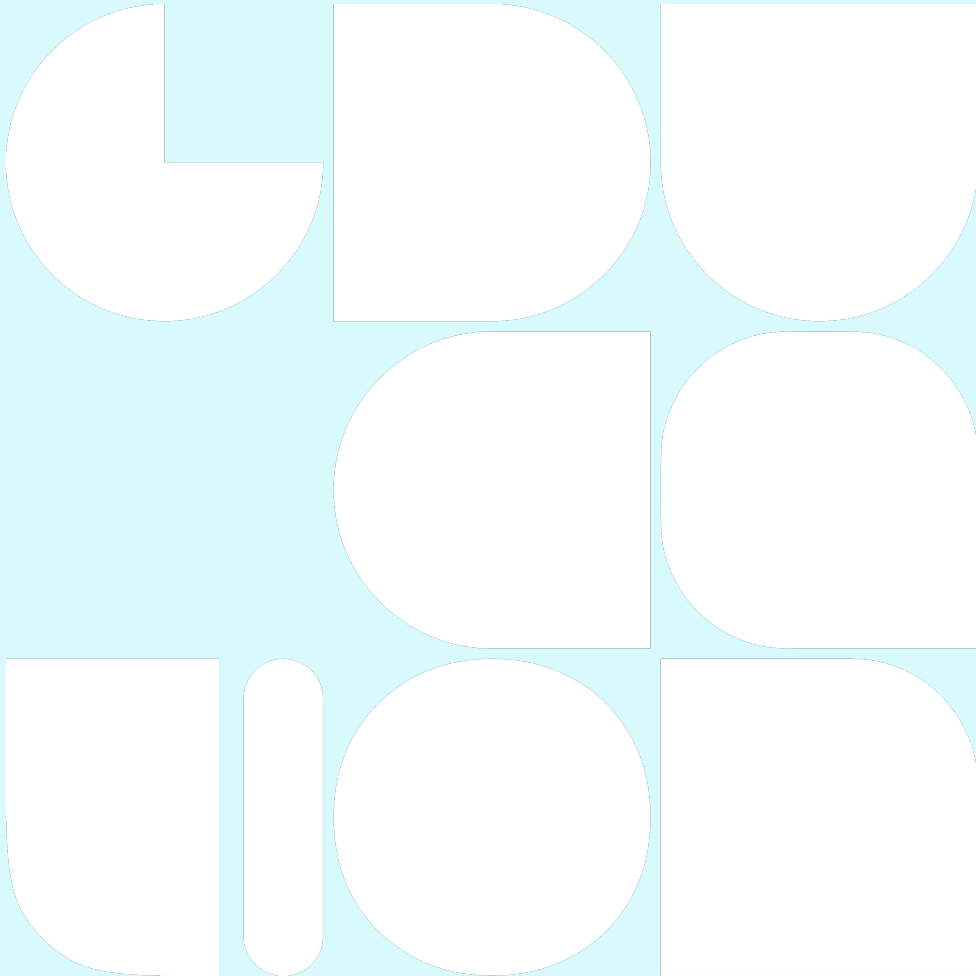
FlashAttention

[FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness]

education

Что еще можете вспомнить в качестве методов оптимизации скорости обучения?

1. Параллелизация
2. Оптимизации обучения
3. Квантизации
4. Оптимизации инференса
5. Фреймворки

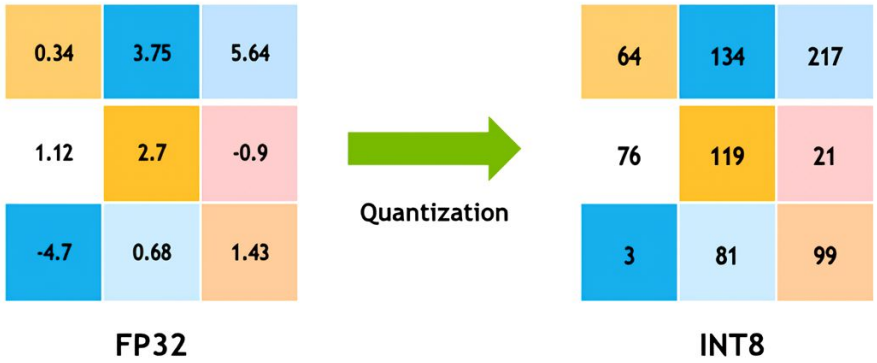


Квантизация

Квантизация — это преобразование весов и активаций модели из 32-битных чисел с плавающей запятой (FP32) в числа с меньшей разрядностью (например, INT8, INT4, FP8 и др.).

Квантизация обычно выполняется после обучения у уже обученной модели и не требует дополнительного обучения, при этом приводит к снижению точности.

При этом можно проводить обучение с учетом квантизации (QAT). В этом случае на каждом шаге происходит эмуляция квантизации. Это позволяет уменьшить погрешность, но при этом увеличивает время обучения



Квантизация

INT8

1

Хороший баланс между производительностью и точностью. Поддерживается большинством аппаратных платформ и библиотек (NVIDIA TensorRT, Intel OpenVINO, PyTorch, ONNX Runtime).

FP8

2

Просто 8 битные плавающие числа. Особо отважные могут в них даже учить, 8 битный Adam довольно надежен

GPTQ

3

Стандарт инференса LLM, квантизация через вычисление обратного Гессиана на трейн датасете.

AWQ

4

Квантует не все веса, а оставляет наиболее важные в более высокой точности (обычной FP16 или BF16)

Q4_K_M

5

Стандарты квантизации в формате GGUF. 4 битные инты + скейлинг + bias для нуля блока.

NVFP4 (он же MXFP4)

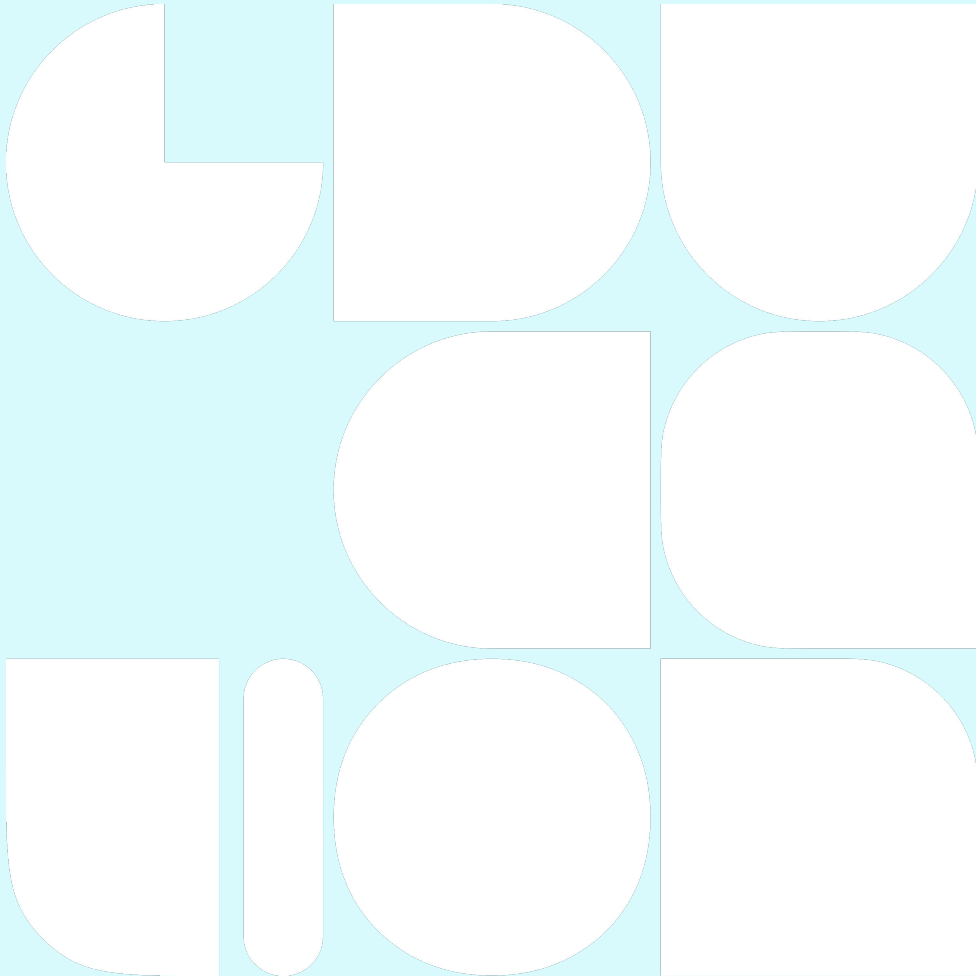
6

Стандарт от Nvidia. 2 стадийная квантизация: все веса делятся на блоки (обычно по 16-32 весов), для каждого блока еще хранится в FP8 scaling factor.

education

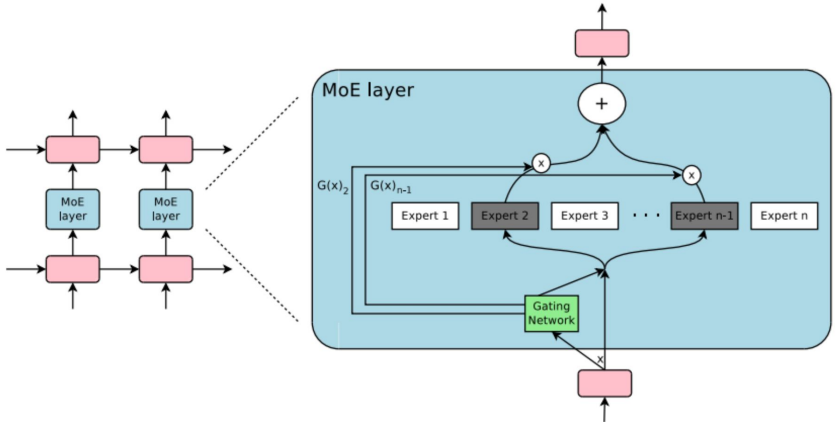
Какие минусы еще есть у квантизаций, кроме ухудшения качества?

1. Параллелизация
2. Оптимизации обучения
3. Квантизации
4. Оптимизации инференса
5. Фреймворки



MOE (2017)

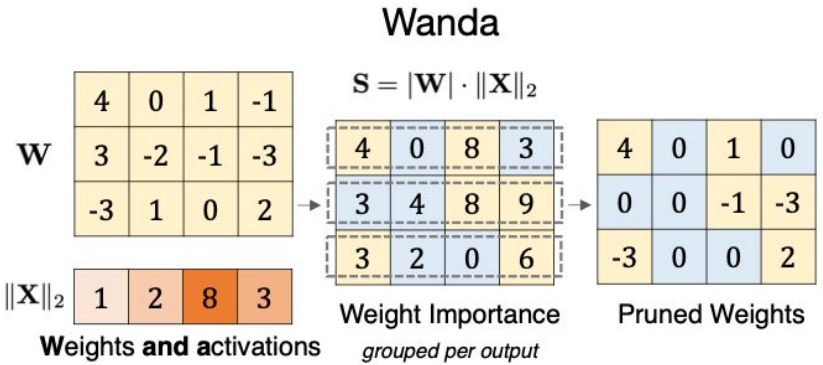
1. Модель содержит множество экспертов на некотором наборе слоев.
2. Для каждого входа специальный gating-механизм выбирает, какие эксперты будут использоваться.
3. Обычно активируется не более 10% экспертов из десятков или сотен.
4. Остальные эксперты "отдыхают"
5. Более простой инференс на нескольких GPU, т.к. передача данных между GPU ограничена выдачей экспертов, количество которых ограниченно.



Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer
Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity

Wanda (2023)

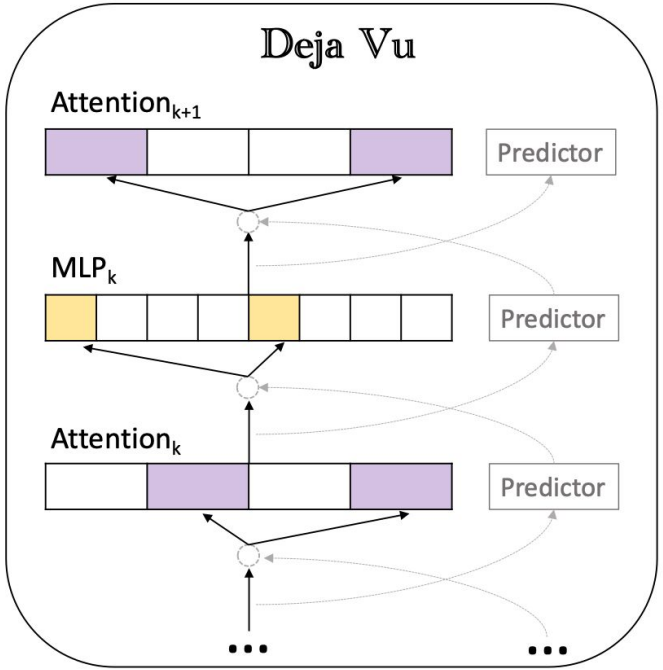
1. Используется информация не только о самих весах, но и об их влиянии на активации (выходы нейронов).
2. Для каждого веса оценивается его важность на основе произведения значения веса и средней величины соответствующей активации.
3. Наиболее "малозначимые" веса (с низким вкладом в активации) обнуляются.



[A Simple and Effective Pruning Approach for Large Language Models]

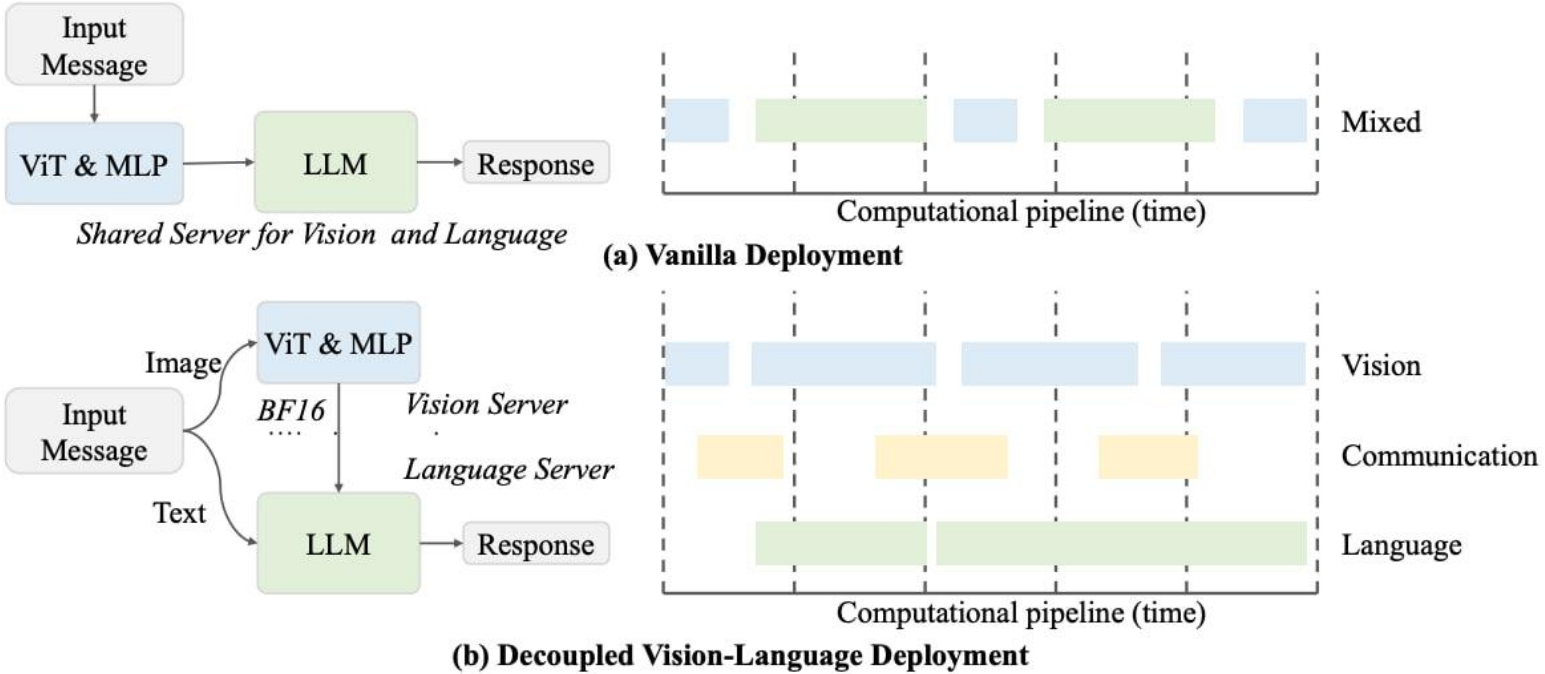
Deja vu (2023)

1. Метод прунинга основанный на использовании небольших подсетей-предикторов, которые обучаются на том же датасете, что и основная сеть.
2. Эти предикторы предсказывают какие веса нужно использовать в конкретном примере.
3. Не удаляет веса, больше похож на inference версию MoE.
4. Пока происходит подсчет результатов текущего слоя считается и предиктор для следующего.
5. На OPT-175 скорость работы увеличена в 6 раз, относительно исходной реализации, при этом результаты на бенчах... улучшились



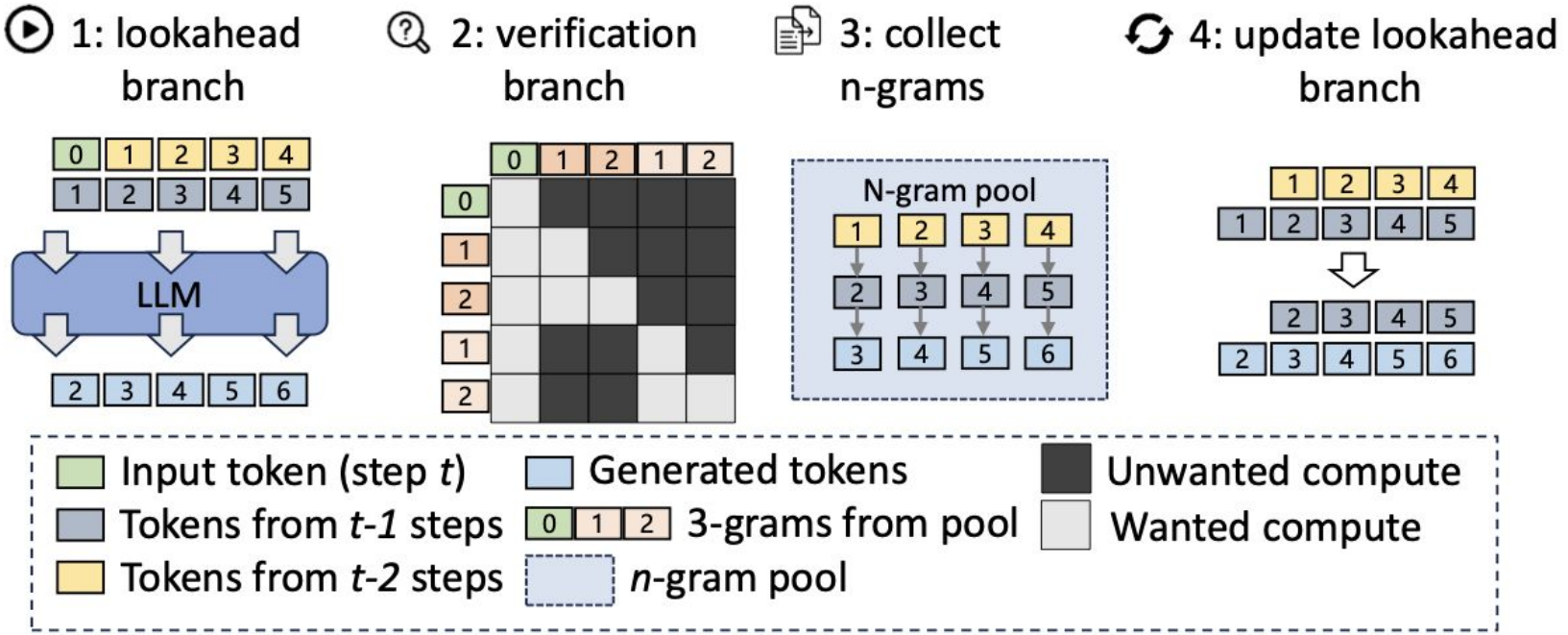
[Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time]

Decoupled Vision-Language Deployment (2025)



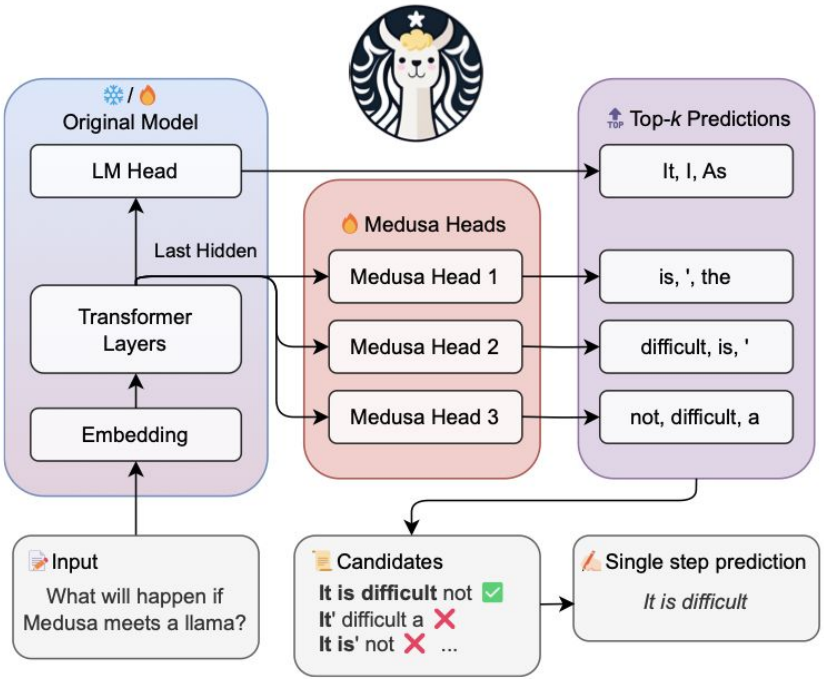
InternVL3.5: Advancing Open-Source Multimodal Models in Versatility, Reasoning, and Efficiency

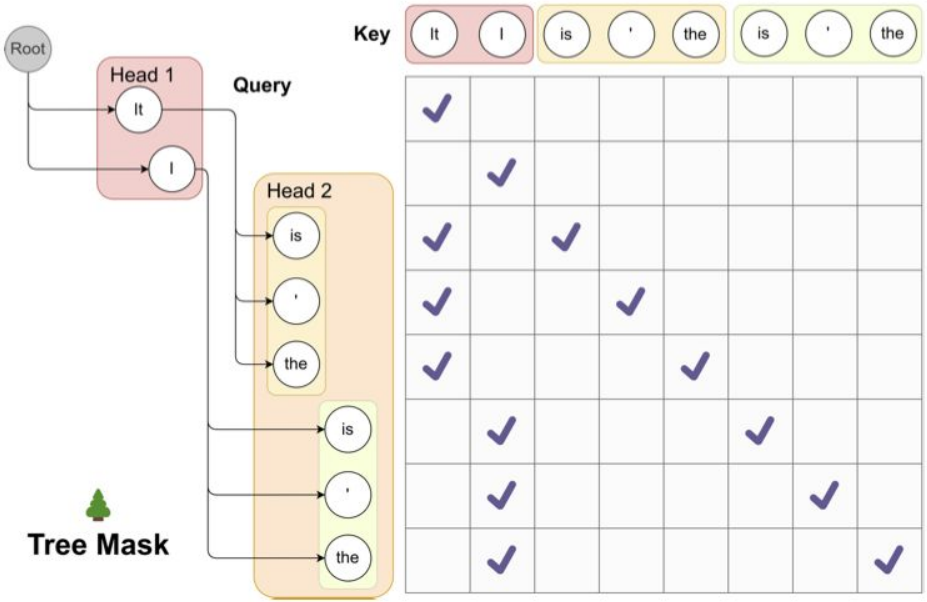
Lookahead decoding (2023)



[Break the Sequential Dependency of LLM Inference Using Lookahead Decoding]

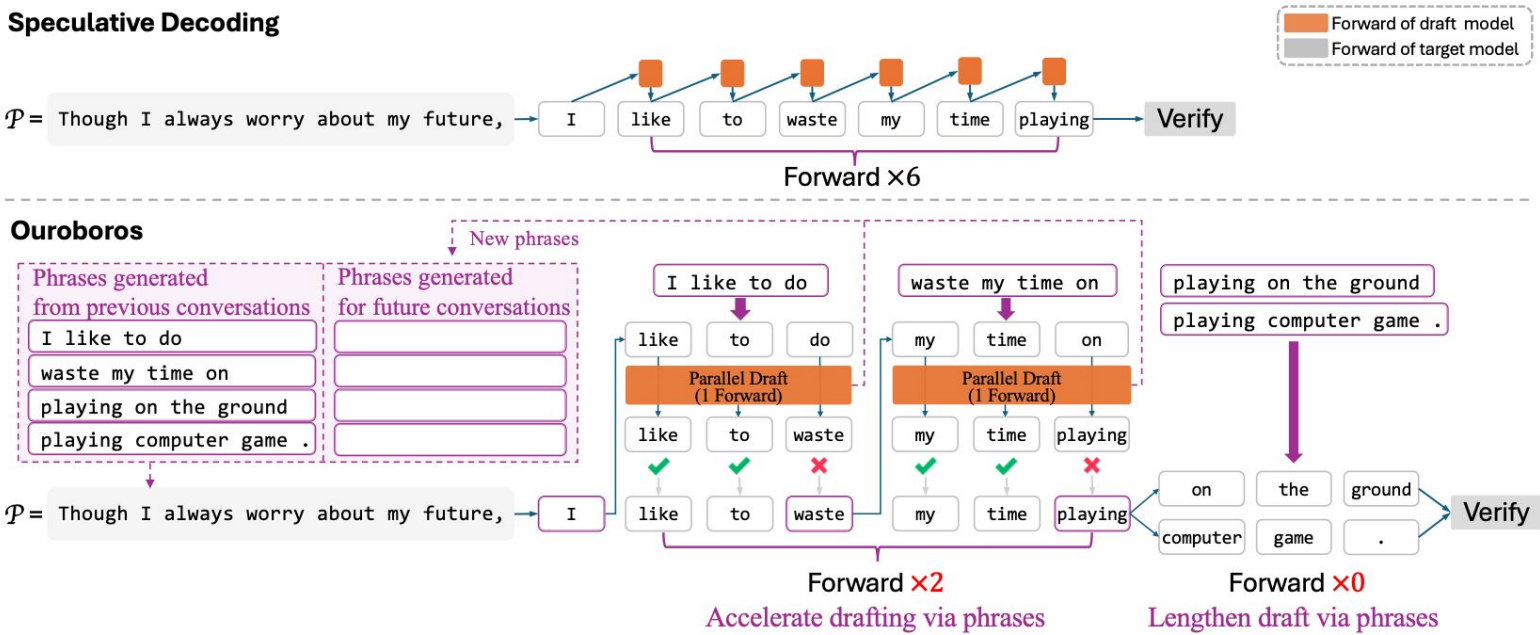
Medusa (2024)





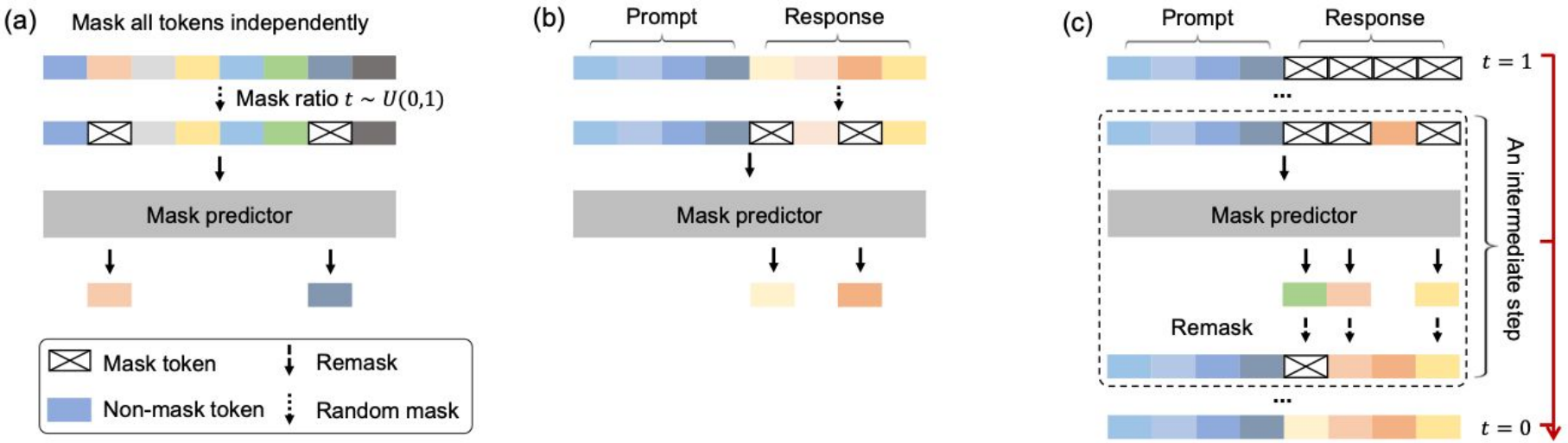
[Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads]

Ouroboros (2024)



[Ouroboros: Generating Longer Drafts Phrase by Phrase for Faster Speculative Decoding]

LLADA (2025)

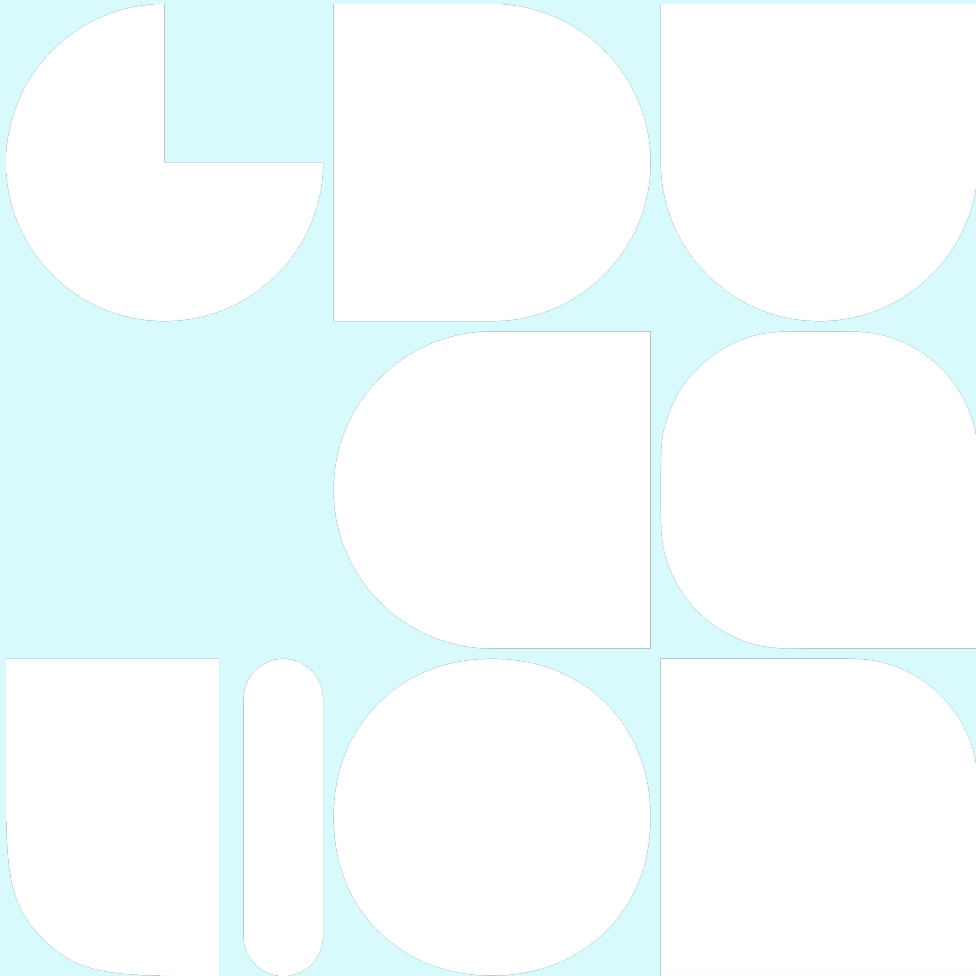


Large Language Diffusion Models

education

Какие направления ускорения инференса вы можете выделить?

1. Параллелизация
2. Оптимизации обучения
3. Квантизации
4. Оптимизации инференса
5. Фреймворки



Deepspeed

DeepSpeed — открытая библиотека от Microsoft для эффективного и масштабируемого обучения и инференса больших нейросетей (особенно языковых моделей). Позволяет обучать модели с сотнями миллиардов параметров на кластерах из тысяч GPU.

Ключевые особенности:

1. ZeRO (Zero Redundancy Optimizer): параллельный оптимизатор
2. Гибридная параллелизация: Поддержка data, tensor, pipeline, sequence и complex parallelism для максимального масштабирования.
3. Оптимизация памяти: Экономия до 10–50x GPU-памяти за счёт ZeRO, активационных чекпоинтов и квантизации.
4. Ускорение инференса: Поддержка INT8/INT4 квантизации, Flash-Attention, оптимизированных коммуникаций.
5. Интеграция с PyTorch и Hugging Face: Простое подключение к существующим пайплайнам.

https://www.deepspeed.ai/

vllm

vLLM — это высокопроизводительная библиотека для ускоренного инференса больших языковых моделей (LLM) на GPU. Разработана для поддержки массового параллельного генерации текстов (massive throughput) и эффективного использования памяти.

Ключевые особенности:

1. PagedAttention: механизм управления памятью для attention, позволяющий эффективно обслуживать тысячи параллельных запросов без перерасхода GPU-памяти.
2. Массовый батчинг:Одновременная генерация тысяч текстов с минимальной задержко.
3. Поддержка популярных моделей: Llama, Falcon, GPT-OSS, Qwen и др.
4. Совместимость с Hugging Face:
   Простая интеграция через стандартный API (transformers, OpenAI API).
5. Квантизация: Поддержка INT8/INT4, AWQ, GPTQ.

https://docs.vllm.ai/en/latest/

bitsandbytes

bitsandbytes — open-source библиотека, разработанная Meta Research, для эффективной квантизации и оптимизации нейросетей на GPU. Позволяет запускать и обучать большие языковые модели (LLM) с использованием INT8, INT4 и NF4 форматов, значительно экономя память и ускоряя вычисления.

Ключевые особенности:

1. INT8/INT4/NF4 квантизация: Снижает требования к памяти и ускоряет инференс и обучение без существенной потери точности.
2. Квантизация весов и градиентов: Поддержка квантизации не только весов, но и градиентов для оптимизации обучения.
3. GPTQ, QLoRA, LORA: Интеграция с современными методами квантизации и дообучения LLM.
4. Совместимость с PyTorch и Hugging Face Transformers: Простое подключение к существующим пайплайнам и моделям.
5. Оптимизированные оптимизаторы: Быстрые реализации Adam, LAMB и других оптимизаторов с поддержкой квантизации.

https://github.com/bitsandbytes-foundation/bitsandbytes

llama.cpp

llama.cpp — это легковесная, высокоэффективная библиотека и CLI-инструмент для запуска больших языковых моделей на обычных CPU и GPU. Написана на голом C/C++ и не требует установки тяжёлых ML-фреймворков.

Ключевые особенности:

1. Запуск LLM на CPU и GPU: Позволяет запускать Llama, Llama 2, Llama 3, Mistral, Phi-2, Qwen, Gemma и другие модели на широком спектре устройств.
2. Широкий спектр поддерживаемых квантизаций: Поддержка форматов от 1.5 до 8 бит
3. Высокая производительность: Оптимизированные вычисления (SIMD, AVX, CUDA, Metal, OpenCL, Vulkan, ROCm).
4. Потоковая генерация: Мгновенный вывод токенов, поддержка многопользовательских и чатовых сценариев.
5. Минимальные зависимости: Не требует Python, PyTorch, CUDA — только компилятор C/C++.

https://github.com/ggml-org/llama.cpp

triton

Triton — открытый язык программирования и компилятор для написания высокоэффективных пользовательских ядер (kernels) на GPU. Позволяет создавать быстрые операции для глубокого обучения, не используя CUDA напрямую, и автоматически оптимизирует код под современные архитектуры.

Ключевые особенности:

1. Python-подобный синтаксис: Позволяет писать GPU-ядра без глубоких знаний CUDA.
2. Автоматические оптимизации: Компилятор Triton сам подбирает параметры для эффективной работы на разных GPU (NVIDIA, AMD).
3. Высокая производительность: Ядра Triton часто превосходят по скорости стандартные реализации PyTorch и TensorFlow.
4. Интеграция с PyTorch: Используется для ускорения операций в PyTorch 2.x (torch.compile, torchinductor).
5. Расширяемость: Можно создавать свои кастомные операции для специфических задач (например, другие версии attention’ов, квантизаций и тд и тп).

https://github.com/triton-lang/triton

education

Какие еще фреймворки ускорения MMLM вы можете вспомнить?

Что мы обсудили на этой лекции?

1. Что можно оптимизировать в MMLM
2. Способы параллелизации
3. Способы ускорения обучения
4. Способы ускорения инференса
5. Квантизации
6. Фреймворки для оптимизации

Спасибо
за внимание!

education