



AICompass

模型对算力等效建模评估工具

一、项目简介

AICompass 是一套面向异构算力场景的智能评估工具，能够在可视化界面根据各类大模型参数及结构，计算其显存及算力需求，智能推荐满足服务指标要求的异构硬件所需配置方案。

同时提供算力及互联带宽测试工具，从大模型推理部署需求出发，搭建跨架构统一表征框架，精准测量异构加速卡性能。

通过大量的测试实验，修正计算公式，显存占用预测误差控制在5%以内，推荐的硬件配置能够满足吞吐量指标要求，助力智算中心实现从“盲目建设”到“按需供给”的转型。

体验链接：<https://siborn.top>



团队成员：季玮晔、许博文

指导老师：杨鹏飞

1.1 赛题完成情况

| 赛题要求 | 完成情况 |
|--|--|
| 支持多种模型类型，模型参数，量化方案，并发量，最大tokens数等负载建模能力 | <input checked="" type="checkbox"/> 支持主流稠密、稀疏、多模态大模型，支持自定义模型结构参数，支持自定义负载及服务指标 |
| 支持多种硬件资源，显存，卡数等资源抽象能力 | <input checked="" type="checkbox"/> 支持国内外异构硬件，包括GPU、TPU、NPU、FPGA等，能够智能推荐硬件配置需求 |
| 建模方案。提供详细的负载、硬件资源的建模方案描述，给出度量依据、建模方法、参考文献、算法依据、计算过程等 | <input checked="" type="checkbox"/> 支持显存占用分析，支持通过算力及互联带宽计算硬件需求，并详细给出计算公式、以及算法依据 |
| 算力需求预测误差：不同测试用例下，度量工具输出的模型算力需求与实际运行真值偏差 | <input checked="" type="checkbox"/> 经过实测验证，在不同输入负载下，各部分显存计算误差不超过5%，推荐硬件配置能够满足吞吐量要求 |
| 代码结构清晰，模块化设计，便于维护和扩展。重要代码注释完整。且必须完全开源 | <input checked="" type="checkbox"/> 网页、算力测试工具、可视化绘图代码全部开源并配有完整注释 |
| 文档详细清晰，包含安装方式及使用方法 | <input checked="" type="checkbox"/> 使用方法见详细本README文档及说明书 |

1.2 核心工作

(1) 可视化平台 ：通过网页部署形式展示评估工具，用户可以通过桌面端或移动端浏览器访问，方便直观。网页支持主流或自定义大模型，支持稠密、稀疏、多模态大模型，以可视化方式展示各部分显存占用情况。选定部署指标及服务指标要求，页面能够给出异构硬件最小部署需求。另外提供推理速度模拟，供服务指标修改参考。

(2) 算力测试工具 ：提供一套针对异构硬件的算力及互联带宽测试工具，通过实验分析，确定大模型推理部署算子及其尺寸，构建统一的算力及互联带宽等效建模能力，解决理论值难以精确预测硬件需求的问题，从而提供准确的硬件方案建议，测试工具代码已开源。

(3) 充分实测验证 ：本项目选取受广泛认可的经典模型，如 `llama\mixtral\llava` 等，对不同的输入负载，测试不同并发量下的模型各部分显存占用，经对比误差小于5%。对某国产加速卡，使用评估工具建议的硬件配置，对 `DeepSeek-R1 671B` 满血版进行测试，能够满足服务指标及吞吐量要求，测试及可视化代码已开源。

| 创新点 | 完成情况 |
|-------------------|------|
| 支持自定义模型、自定义硬件 | ✓ |
| 根据模型类型精确划分显存占比 | ✓ |
| 使用经验因子控制额外开销计算 | ✓ |
| 通过算力评估工具实测硬件真实算力 | ✓ |
| 通过模型部署实测确定显存计算准确性 | ✓ |
| 通过SLO测试确定推荐卡数达标情况 | ✓ |

1.3 对比其他工具的核心优势

| 能力 | AICompass | 市面现有工具 |
|----------|----------------------------|-----------|
| 自定义模型 | ✓ 支持任意参数/注意力机制 | ✗ 仅限固定模型 |
| 多模态与稀疏模型 | ✓ Dense/MoE/Multimodal 全覆盖 | ✗ 仅稠密模型 |
| 硬件兼容性 | ✓ 国内外异构加速硬件 + 自定义硬件 | ✗ 有限硬件支持 |
| 显存计算粒度 | ✓ 权重/KV Cache/激活值/量化分项 | ✗ 简单总量估算 |
| 延迟目标驱动部署 | ✓ 基于 TTFT/TPOT 动态推荐卡数 | ✗ 仅显存容量计算 |
| 实测验证 | ✓ 开源测试程序 + ≤5% 误差 | ✗ 无验证机制 |

1.4 应用价值

AICompass 服务于大模型推理部署决策，解决开发者三大痛点，助力智算中心建设：

1. 显存估算不准 → 精细模型公式 + 多场景实测验证；
2. 硬件选型盲目 → 跨架构卡数推荐 + SLO 约束；
3. 部署成本过高 → 量化方案支持 + 推理框架优化建议。

二、项目结构

```
AICompass/
├── benchmark/
|   ├── DenseBench.py          # AI模型算力与显存评估工具
|   ├── MoeBench.py            # 性能基准测试模块
|   ├── MultimodalBench.py     # 密集型模型(如LLaMA)基准测试
|   └── PerfTest.py            # 混合专家模型(MoE)基准测试
├── README.md                  # 多模态模型基准测试
└── visualize/
    ├── Densevis.py            # 硬件性能测试(GEMM、内存带宽等)
    ├── MoeVis.py               # 项目说明文档
    └── Multimodalvis.py       # 数据可视化模块
└── web/
    ├── data/
    |   ├── gpu.json             # 密集型模型测试结果可视化
    |   ├── logo.png              # MoE模型测试结果可视化
    |   └── model.json            # 多模态模型测试结果可视化
    ├── index.html                # 多模态模型测试结果可视化
    ├── script.js                 # web界面模块
    └── style.css                  # 静态数据文件
                                    # GPU硬件参数配置数据
                                    # 项目logo图标
                                    # 预设模型参数配置数据
                                    # 主页面HTML文件
                                    # 前端JavaScript逻辑
                                    # 页面样式表
```

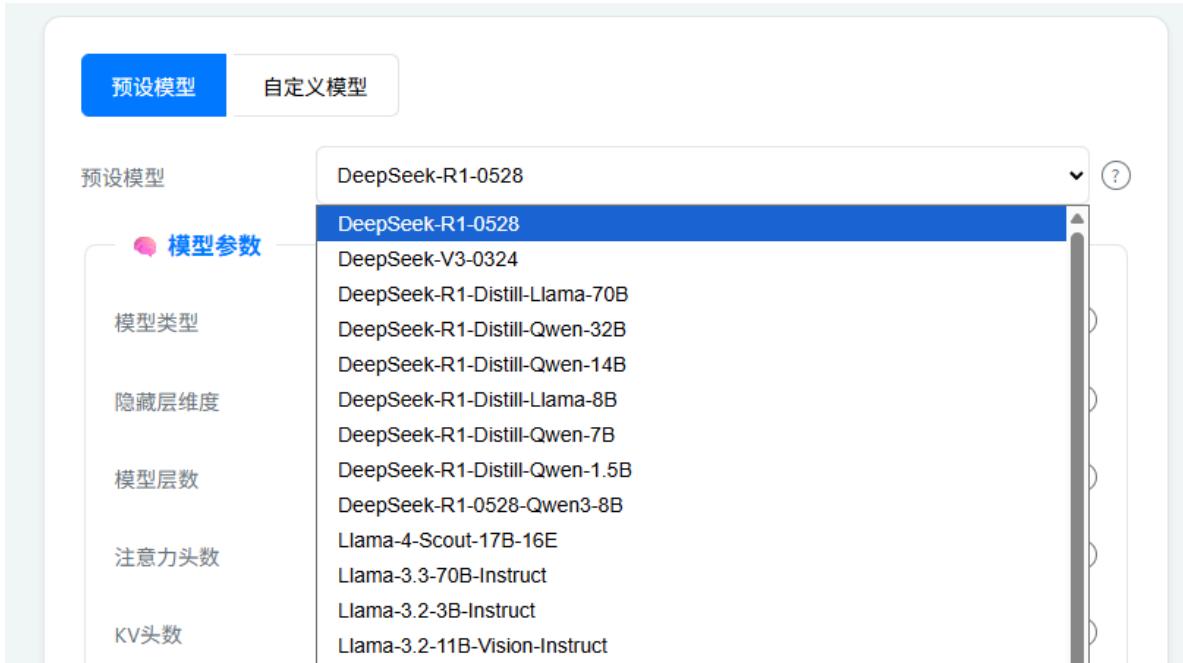
三、使用方法

- 打开 index.html 文件 (建议使用现代浏览器如 Chrome、Firefox 等)



- 选择模型配置方式:

- 预设模型: 从下拉菜单选择主流模型 (如 DeepSeek、Llama 等)



- 自定义模型：手动输入模型参数（支持 Dense、MoE、多模态三种类型）

预设模型

自定义模型

模型参数

模型类型: Dense(稠密)

隐藏层维度: 例如: 4096

模型层数: 例如: 32

注意力头数: 例如: 32

KV头数: 例如: 8

词汇表大小: 例如: 32000

最大上下文长度: 32768

总/基础参数(B): 例如: 8 (十亿)

3. 调整推理配置：

- 设置量化方案（权重、KV Cache、激活值）

⚙ 推理配置

| | | |
|---------------|------|---|
| 权重 量化方案 | FP16 | ? |
| KV Cache 量化方案 | FP16 | ? |
| 激活值 量化方案 | FP16 | ? |

- 调整输入/输出长度和并发量

输入长度: **1024 / 163840**

A horizontal slider for adjusting the input length. The scale ranges from 1 to 164K. The current value is set at 1024.

输出长度: **1024 / 163840**

A horizontal slider for adjusting the output length. The scale ranges from 1 to 164K. The current value is set at 1024.

并发量(Batch Size): **4**

A horizontal slider for adjusting the batch size. The scale ranges from 1 to 256. The current value is set at 4.

额外开销比例

0.03

?

- 配置服务指标 (TTFT、TPOT)

📈 服务指标

TTFT(ms) **500** ?

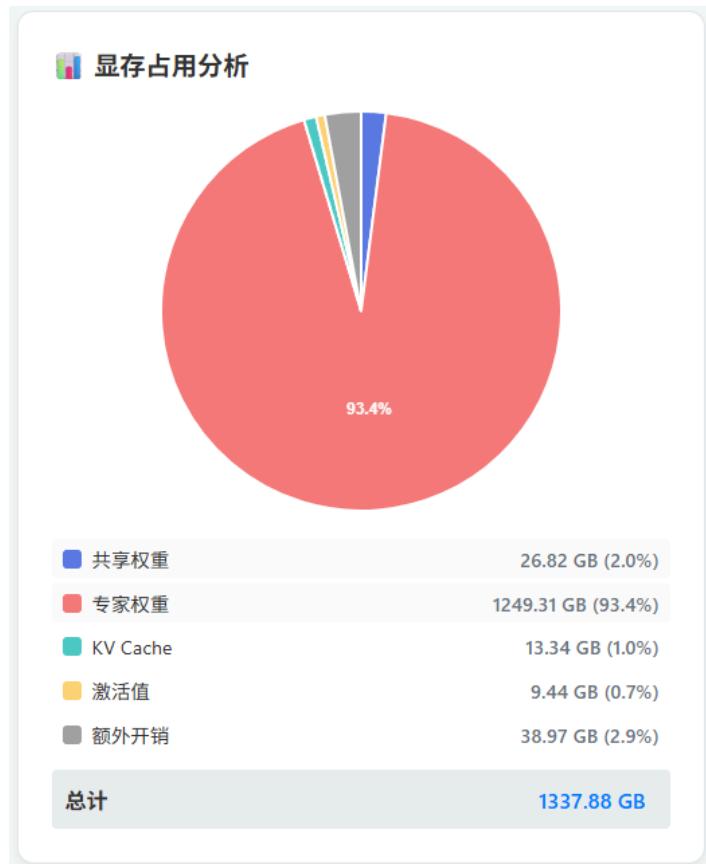
TPOT(ms) **50** ?

吞吐量

⚡ **20 Tokens/s**

?

4. 查看显存占用分析：实时显示内存分布饼图



5. 获取硬件推荐:

- 推荐配置: 查看适合的 GPU 型号和数量

推荐配置 自定义配置

推荐硬件

异构资源利用率 ?

| 厂商 | 型号 | 架构 | 显存 | 预估卡数 |
|--------|---------------------|-------|--------|------|
| NVIDIA | A100 PCIe | GPU | 80 GB | 17 |
| NVIDIA | H100 PCIe | GPU | 80 GB | 17 |
| AMD | Instinct MI300X | GPU | 192 GB | 7 |
| Huawei | Atlas 300I Duo 96GB | NPU | 96 GB | 14 |
| 壁仞科技 | BR100 | GPGPU | 64 GB | 21 |

或指定硬件: ▾

👉 需要 14 张

- 自定义配置: 输入硬件规格计算所需卡数

The screenshot shows the 'Custom Configuration' tab selected in the header. Below it, the title 'Custom Hardware' is displayed. A table lists various hardware specifications with their current values:

| | |
|----------------|------|
| 显存(GB) | 80 |
| 显存带宽(GB/s) | 2039 |
| FP16算力(TFLOPS) | 312 |
| BF16算力(TFLOPS) | 312 |
| FP8算力(TFLOPS) | 0 |
| INT8算力(TOPS) | 624 |
| INT4算力(TOPS) | 1248 |
| 互联带宽(GB/s) | 600 |
| GPU利用率 | 0.8 |

A large blue button labeled '计算卡数' (Calculate Card Count) is centered below the table. At the bottom, a message states '预估需要卡数: 17 张' (Estimated card count: 17 cards).

6. 可选功能：

- 启动推理速度模拟：观察实时 Token 生成过程



四、变量说明

模型参数

| 中文名称 | 英文名称 | 英文缩写 | 说明 |
|-------|-------------------------|------|---------------------------|
| 隐藏层维度 | Hidden Size | H | 模型中隐藏层的维度大小 |
| 模型层数 | Number of Hidden Layers | L | 模型中 Transformer Block 的层数 |

| 中文名称 | 英文名称 | 英文缩写 | 说明 |
|----------|-----------------------------|----------|-----------------------------|
| 注意力头数 | Number of Attention Heads | H_attn | 多头注意力机制中的头的数量 |
| KV 头数 | Number of Key-Value Heads | H_kv | 用于 GQA/MQA 的 Key/Value 头的数量 |
| 词汇表大小 | Vocabulary Size | V | 模型词汇表的大小 |
| 头维度 | Head Dimension | D_h | 每个注意力头的维度，等于隐藏层维度/注意力头数 |
| 总参数量 | Total Parameters | P_total | 模型的总参数量(十亿) |
| 共享参数量 | Shared Parameters | P_shared | MoE 模型中的共享参数量 |
| 单个专家参数量 | Parameters Per Expert | P_expert | 每个专家的参数量(十亿) |
| 专家中间层维度 | Intermediate Size | I | MoE 专家网络中 FFN 的中间层维度 |
| 专家总数 | Number of Local Experts | E | 每个 MoE 层包含的专家总数 |
| 激活专家数 | Number of Experts Per Token | E_active | 每个 Token 激活的专家数量 |
| LLM基础参数量 | LLM Base Parameters | P_base | 多模态模型中 LLM 部分的参数量 |
| 视觉模态参数量 | Vision Modal Parameters | P_vision | 视觉模态的总参数量 |
| 音频模态参数量 | Audio Modal Parameters | P_audio | 音频模态的总参数量 |

推理配置

| 中文名称 | 英文名称 | 英文缩写 | 说明 |
|-----------|------------------------------------|----------|-----------------------|
| 权重量化精度 | Weight Quantization Precision | Q_weight | 模型权重的量化精度(字节/参数) |
| KV 缓存量化精度 | KV Cache Quantization Precision | Q_kv | KV Cache 的量化精度(字节/元素) |
| 激活值量化精度 | Activations Quantization Precision | Q_act | 中间激活值的量化精度(字节/元素) |
| 输入长度 | Input Length | S_in | 输入序列的长度(Token 数) |
| 输出长度 | Output Length | S_out | 输出序列的长度(Token 数) |

| 中文名称 | 英文名称 | 英文缩写 | 说明 |
|-------------|--------------------|------------|-------------------------|
| 序列长度 | Sequence Length | S | 输入长度+输出长度 |
| 批处理大小 | Batch Size | B | 并发处理的请求数量 |
| 额外开销比例 | Overhead Ratio | R_overhead | 用于估算碎片、CUDA 上下文等额外开销的比例 |
| 图像输入尺寸 | Image Input Size | I_size | 输入图像的分辨率(像素) |
| 图像 Patch 尺寸 | Patch Size | P_size | 图像切块(Patch)的大小(像素) |
| 音频输入长度 | Audio Input Length | A_len | 输入音频的时长(秒) |
| 音频采样率 | Audio Sample Rate | A_rate | 音频采样率(Hz) |

服务指标

| 中文名称 | 英文名称 | 英文缩写 | 说明 |
|------------|-----------------------|------|---|
| 首 Token 时间 | Time To First Token | TTFT | 接收请求后，生成首个 Token 的时间(ms)，主要反映 Prefill 阶段性能 |
| 单 Token 时间 | Time Per Output Token | TPOT | 生成后续每个 Token 的平均时间(ms)，主要反映 Decode 阶段性能 |
| 吞吐量 | Throughput | T | 系统每秒能够生成的 Token 总数，由 TPOT 计算得到 ($1/TPOT \times 1000$) |

计算因子

| 中文名称 | 英文名称 | 英文缩写 | 说明 |
|-------------|------------------------------|--------------|---------------------------|
| 稠密激活因子 | Dense Activation Factor | F_act | Dense 模型激活值因子 |
| LLM 激活因子 | LLM Activation Factor | F_llm | 多模态中 LLM 部分激活值因子 |
| 共享激活因子 | MoE Shared Activation Factor | F_shared | MoE 共享部分激活值因子 |
| 专家激活因子 | MoE Expert Activation Factor | F_expert | MoE 专家部分激活值因子 |
| 视觉激活因子 | Vision Activation Factor | F_vision_act | 视觉模态激活值因子 |
| 音频激活因子 | Audio Activation Factor | F_audio_act | 音频模态激活值因子 |
| 视觉 Token 因子 | Vision Token Factor | F_vision | 视觉 token 对 KV Cache 的影响因子 |

| 中文名称 | 英文名称 | 英文缩写 | 说明 |
|-------------|-------------------------|----------|------------------------------|
| 音频 Token 因子 | Audio Token Factor | F_audio | 音频 token 对 KV Cache 的影响因子 |
| 基础 FLOPs 系数 | Base FLOPs Coefficient | F_base | 每 Token 的基础计算量系数 |
| 模型规模系数 | Model Scale Factor | S_model | 基于隐藏层大小和层数的模型规模系数 |
| 注意力效率系数 | Attention Efficiency | E_attn | 注意力机制效率系数 |
| 视觉计算系数 | Vision Computing Factor | F_vision | 视觉模态计算系数 |
| 音频计算系数 | Audio Computing Factor | F_audio | 音频模态计算系数 |
| MoE 路由开销 | MoE Routing Factor | F_moe | MoE 模型路由开销系数 |
| 多模态处理开销 | Multimodal Factor | F_mm | 多模态处理额外开销系数 |
| 批处理效率 | Batch Efficiency | E_batch | 批处理大小对计算效率的影响 |
| 量化精度系数 | Quantization Factor | F_quant | 量化精度对算力需求的影响系数 |
| 全局校准因子 | Global Calibration | G_cal | 基于实际部署经验的全局校准因子 |
| 批处理大小 | Batch Size | B | 并发处理的请求数量 |
| 输入长度 | Input Length | L_in | 输入序列的长度(Token数) |
| 序列总长度 | Sequence Length | L_seq | 输入长度+输出长度 |
| 吞吐量 | Throughput | T | 系统每秒能够生成的 Token 总数(tokens/s) |

硬件参数

| 中文名称 | 英文名称 | 英文缩写 | 说明 |
|---------|------------------------|--------|-------------------------|
| 显存大小 | VRAM Size | M_gpu | GPU 显存大小(GB) |
| 显存带宽 | Memory Bandwidth | BW_mem | GPU 显存带宽(GB/s) |
| FP16 算力 | FP16 Computing Power | C_fp16 | GPU 的 FP16 算力(TFLOPS) |
| BF16 算力 | BF16 Computing Power | C_bf16 | GPU 的 BF16 算力(TFLOPS) |
| FP8 算力 | FP8 Computing Power | C_fp8 | GPU 的 FP8 算力(TFLOPS) |
| INT8 算力 | INT8 Computing Power | C_int8 | GPU 的 INT8 算力(TOPS) |
| INT4 算力 | INT4 Computing Power | C_int4 | GPU 的 INT4 算力(TOPS) |
| 互联带宽 | Interconnect Bandwidth | BW_ic | GPU 间互联带宽(GB/s) |
| GPU利用率 | GPU Utilization | U_gpu | GPU 计算资源的实际利用率(0.1-1.0) |

五、计算方法

显存占用计算

稠密模型(Dense)显存计算

稠密模型的显存占用主要包括模型权重、KV缓存、激活值和额外开销四部分：

模型权重显存：

$$M_{weights} = P_{total} \times Q_{weight}$$

KV缓存显存：

$$M_{kv} = B \times S \times L \times 2 \times H_{kv} \times D_h \times Q_{kv}$$

激活值显存：

$$M_{act} = B \times S \times H \times F_{act} \times Q_{act}$$

额外开销显存：

$$M_{overhead} = (M_{weights} + M_{kv} + M_{act}) \times R_{overhead}$$

总显存：

$$M_{total} = M_{weights} + M_{kv} + M_{act} + M_{overhead}$$

混合专家模型(MoE)显存计算

MoE模型的显存占用包括共享权重、专家权重、KV缓存、激活值和额外开销：

共享权重显存：

$$M_{shared} = P_{shared} \times Q_{weight}$$

专家权重显存：

$$M_{expert} = P_{expert} \times E \times Q_{weight}$$

KV缓存显存：与Dense模型相同

激活值显存：

$$M_{act_shared} = B \times S \times H \times F_{shared} \times Q_{act}$$

$$M_{act_expert} = B \times S \times E_{active} \times I \times F_{expert} \times Q_{act}$$

$$M_{act} = M_{act_shared} + M_{act_expert}$$

额外开销显存：

$$M_{overhead} = (M_{shared} + M_{expert} + M_{kv} + M_{act}) \times R_{overhead}$$

总显存：

$$M_{total} = M_{shared} + M_{expert} + M_{kv} + M_{act} + M_{overhead}$$

多模态模型(Multimodal)显存计算

多模态模型的显存占用包括基础LLM权重、视觉模态权重、音频模态权重、KV缓存、激活值和额外开销：

基础LLM权重显存：

$$M_{base} = P_{base} \times Q_{weight}$$

视觉模态权重显存：

$$M_{vision} = P_{vision} \times Q_{weight}$$

音频模态权重显存：

$$M_{audio} = P_{audio} \times Q_{weight}$$

KV缓存显存：

$$S_{total} = S_{text} + S_{image} \times F_{vision} + S_{audio} \times F_{audio}$$

$$M_{kv} = B \times S_{total} \times L \times 2 \times H_{kv} \times D_h \times Q_{kv}$$

激活值显存：

$$M_{act_llm} = B \times S_{text} \times H \times F_{llm} \times Q_{act}$$

$$M_{act_vision} = B \times S_{image} \times H \times F_{vision_act} \times Q_{act}$$

$$M_{act_audio} = B \times S_{audio} \times H \times F_{audio_act} \times Q_{act}$$

$$M_{act} = M_{act_llm} + M_{act_vision} + M_{act_audio}$$

额外开销显存：

$$M_{overhead} = (M_{base} + M_{vision} + M_{audio} + M_{kv} + M_{act}) \times R_{overhead}$$

总显存：

$$M_{total} = M_{base} + M_{vision} + M_{audio} + M_{kv} + M_{act} + M_{overhead}$$

计算能力需求评估

稠密模型(Dense)算力计算

$$C_{prefill} = \frac{F_{base} \times S_{model} \times L_{in} \times P_{total} \times (1/E_{attn}) \times B}{TTFT}$$

$$C_{token} = F_{base} \times S_{model} \times (1 + 0.1 \log_{10}(L_{seq})) \times P_{total} \times (1/E_{attn}) \times 0.5 \times B$$

$$C_{decode} = C_{token} \times T$$

$$C = \max(C_{prefill}, C_{decode}) \times F_{quant} \times E_{batch} \times G_{cal}$$

混合专家模型(MoE)算力计算

$$P_{effective} = P_{shared} + P_{expert} \times R_{expert}$$

$$C_{prefill} = \frac{F_{base} \times S_{model} \times L_{in} \times P_{effective} \times (1/E_{attn}) \times B \times F_{moe}}{TTFT}$$

$$C_{token} = F_{base} \times S_{model} \times (1 + 0.1 \log_{10}(L_{seq})) \times P_{effective} \times (1/E_{attn}) \times 0.5 \times B \times F_{moe}$$

$$C_{decode} = C_{token} \times T$$

$$C = \max(C_{prefill}, C_{decode}) \times F_{quant} \times E_{batch} \times G_{cal}$$

多模态模型(Multimodal)算力计算

$$C_{prefill} = \frac{F_{base} \times S_{model} \times L_{in} \times P_{base} \times (1/E_{attn}) \times B \times F_{mm}}{TTFT}$$

$$C_{token} = F_{base} \times S_{model} \times (1 + 0.1 \log_{10}(L_{seq})) \times P_{base} \times (1/E_{attn}) \times 0.5 \times B$$

$$C_{decode} = C_{token} \times T$$

$$C_{vision} = P_{vision} \times F_{vision} \times B$$

$$C_{audio} = P_{audio} \times F_{audio} \times B$$

$$C_{modal} = C_{vision} + C_{audio}$$

$$C = (\max(C_{prefill}, C_{decode}) + C_{modal}) \times F_{quant} \times E_{batch} \times G_{cal}$$

硬件需求计算

显卡数量计算

$$N_{mem} = \lceil M_{total} / M_{gpu} \rceil$$

$$N_{comp} = \lceil C / (C_{gpu} \times U_{gpu}) \rceil$$

$$N = \max(N_{mem}, N_{comp})$$

六、测试部分

DenseBench

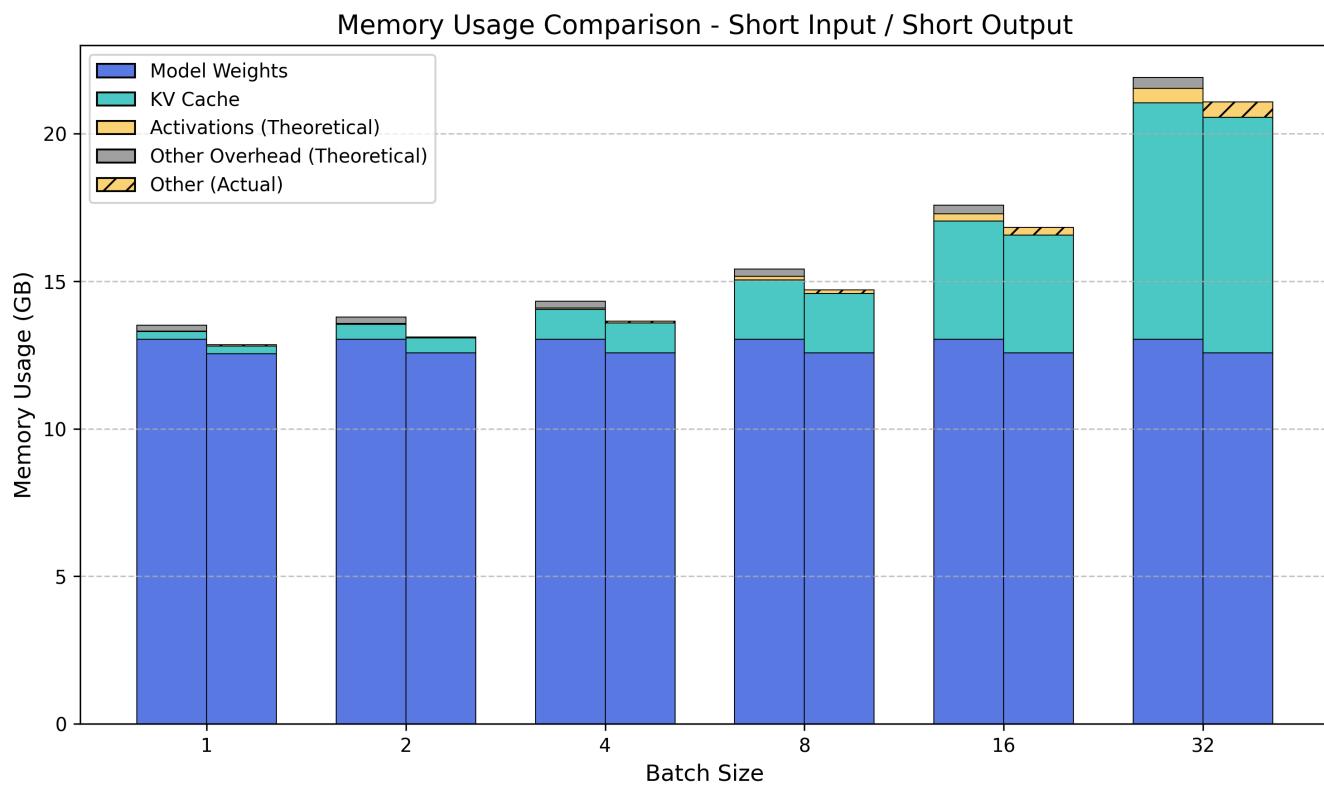
在本项目的 Dense 模型显存测试中，通过理论建模与实测验证的双重手段对资源需求进行全面评估。测试框架基于配置文件解析模型的隐藏层维度、注意力头数、网络层数等结构参数，结合量化精度参数，构建显存消耗的模型。该模型将显存占用解耦为权重存储、KV缓存、激活值及系统开销四个组件：权重存储规模由参数总量与量化精度决定，KV 缓存显存与批处理规模、总序列长度、注意力头维度及网络层数正相关，激活值显存通过引入每 token 的激活因子表征中间数据增量，系统级缓存等未计量项则通过预设比例系数补偿。

实际测试阶段通过构造指定长度的输入文本生成批量化的 GPU 张量，分阶段记录显存变化。模型初始化时获取基础权重加载后的显存基线，完整推理过程中动态监测多卡设备的聚合显存峰值。测试代码调用 `torch.cuda.memory_allocated` 接口实现显存数据的跨卡采集，同步对 KV cache 张量进行空间计算。每次推理后执行显存清理与垃圾回收，确保测试环境的一致性。

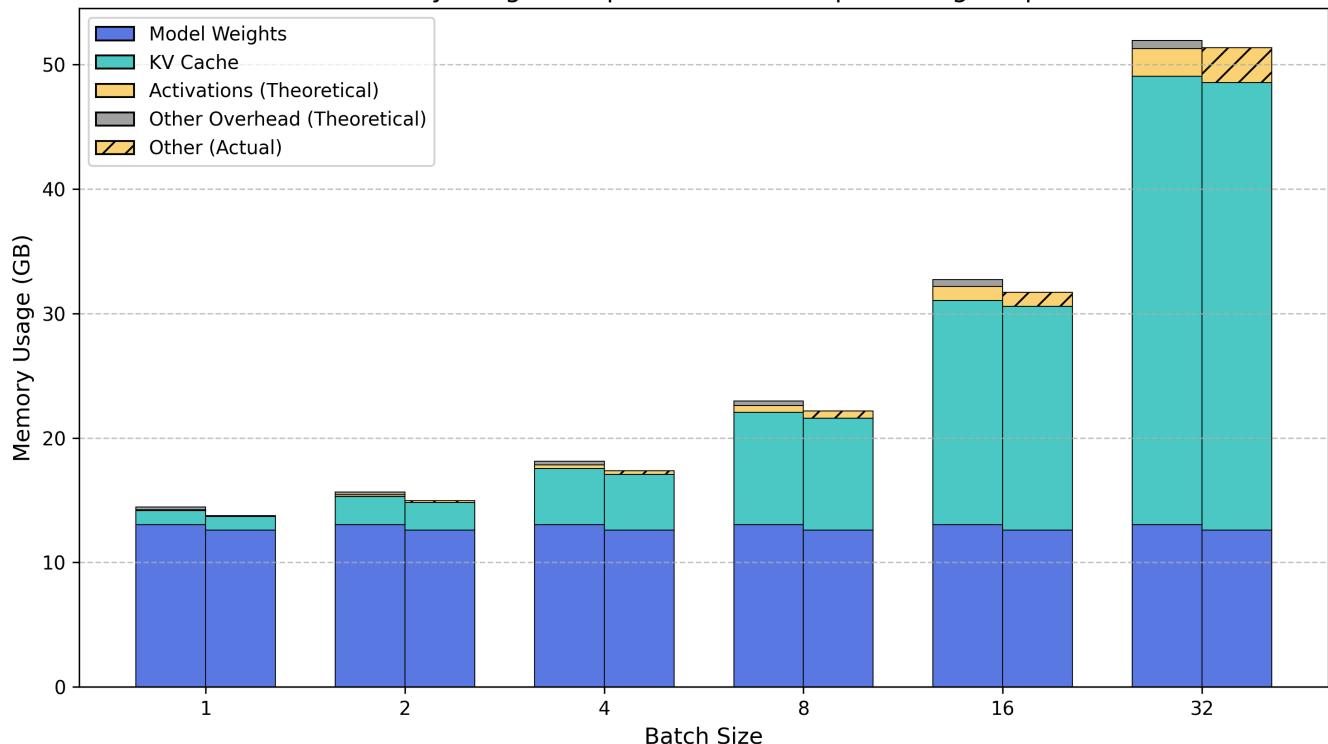
测试方案设定了 16 组参数组合，涵盖批处理规模按 2 的幂次从 1 到 32 的梯度扩展，以及输入/输出序列长度在 256 至 2048 tokens 区间的四种典型组合模式（短输入短输出、短输入长输出、长输入短输出、长输入长输出）。这种多维参数空间设计可系统性揭示批处理规模与序列长度对显存占用的耦合效应，为不同推理场景提供资源消耗基线。

执行过程中测试程序自动生成 CSV 格式报告，包含推理耗时、吞吐量、理论/实测显存分解数据。通过比对权重与 KV 缓存的实际值与理论计算值，可校验估算公式的准确性。当激活值显存出现显著偏差时，差异数据将为经验性激活因子优化提供依据，增强理论模型的泛化能力。最终测试报告不仅输出各场景的显存绝对值，更通过组件级资源占比分析，为分布式训练模型切分、推理服务批处理规模优化等决策提供量化支撑。

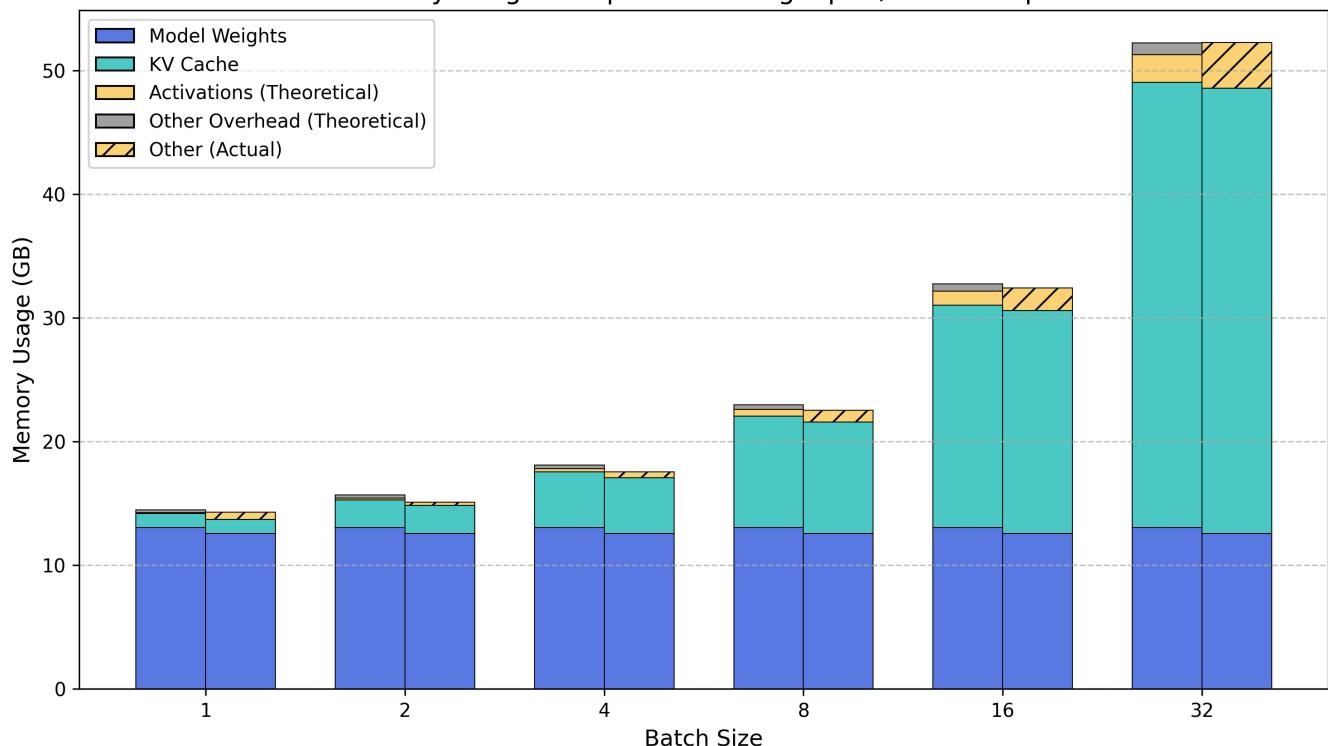
针对 `11ama2-7B/13B` 模型的实测数据显示，理论预估与实测显存占用的误差控制在 5% 以内，验证了计算模型的有效性。



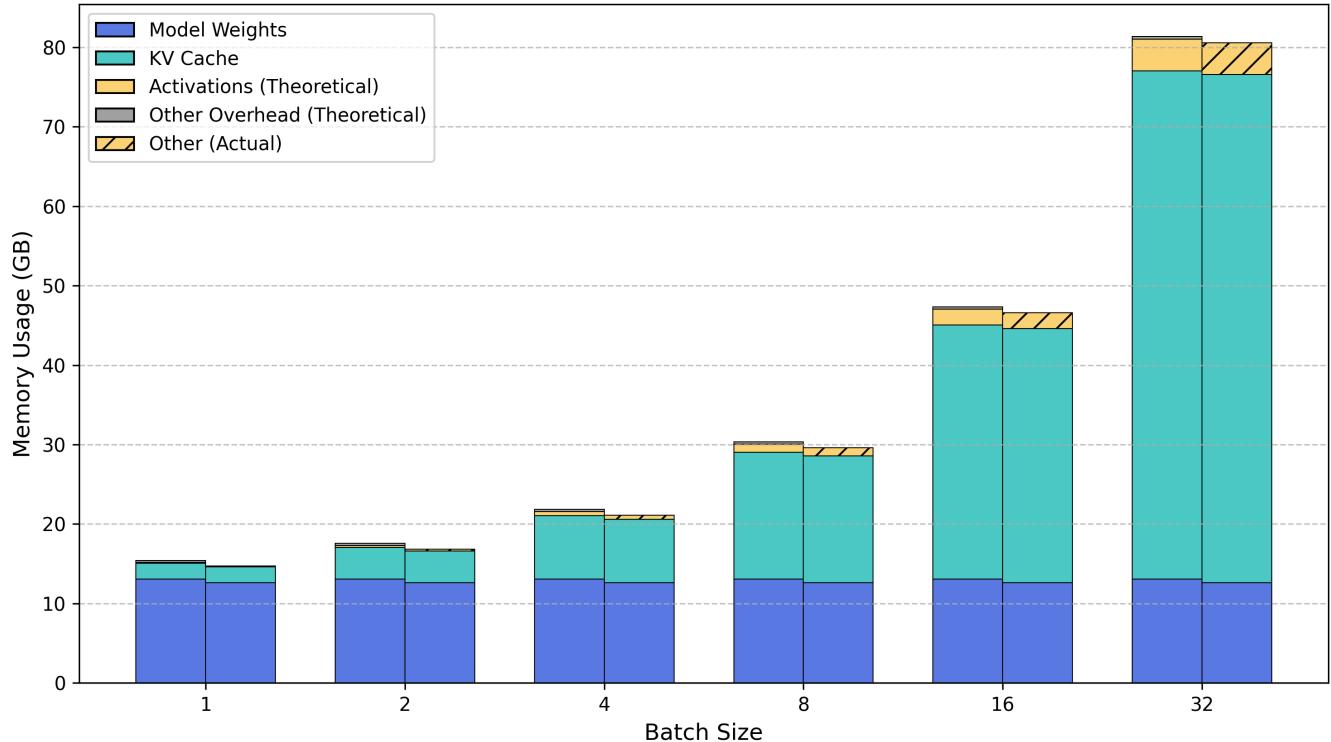
Memory Usage Comparison - Short Input / Long Output



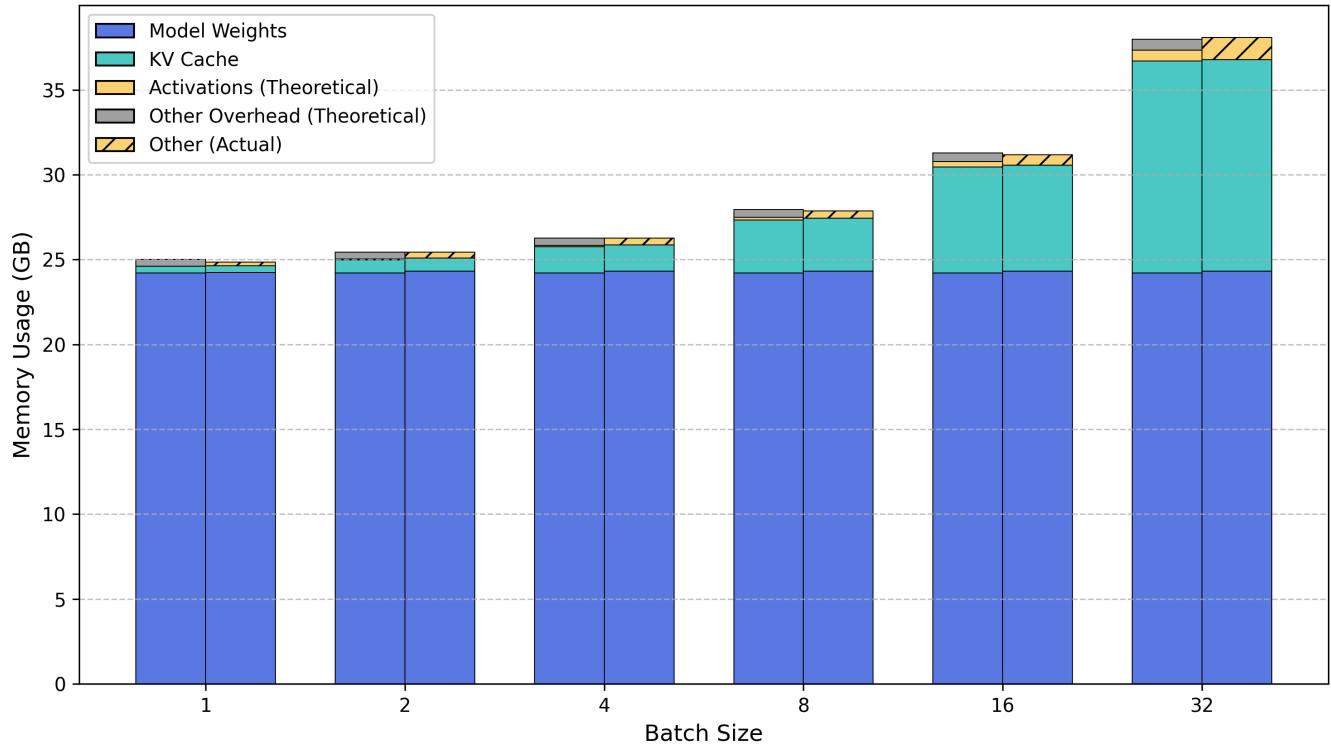
Memory Usage Comparison - Long Input / Short Output



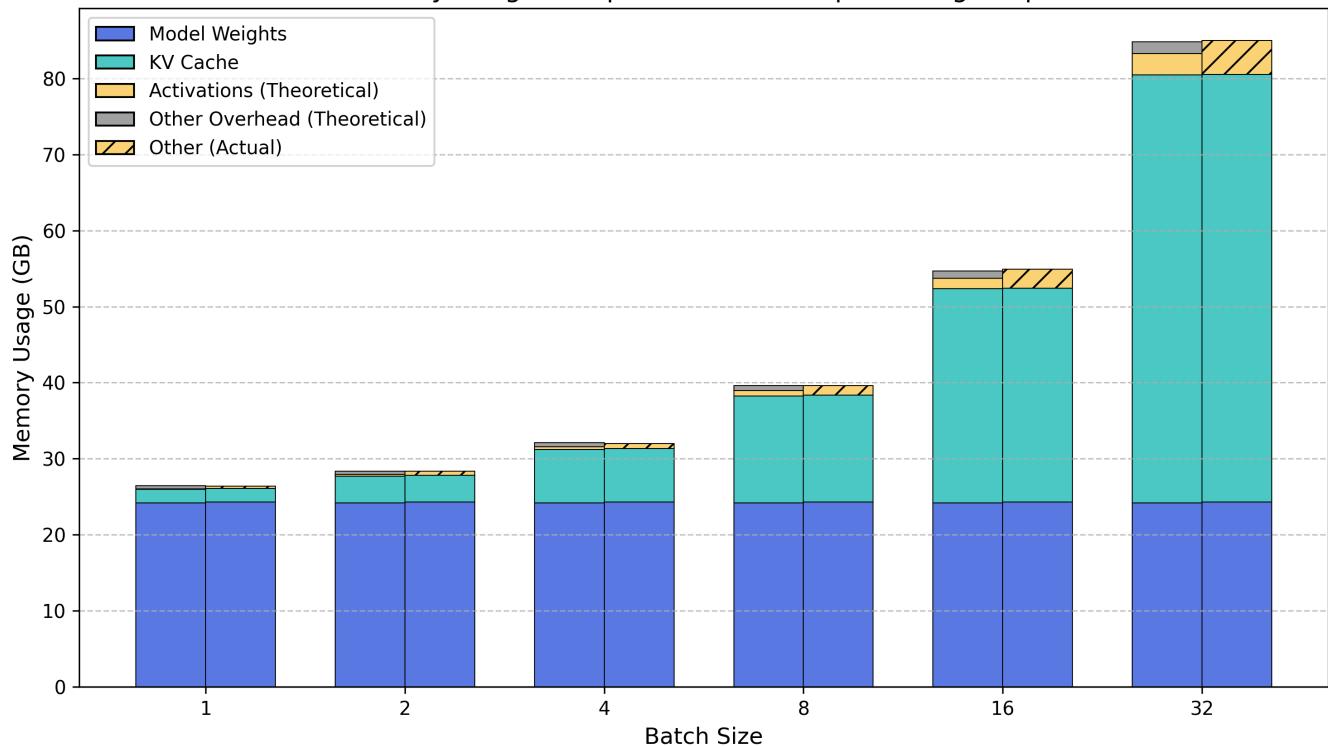
Memory Usage Comparison - Long Input / Long Output



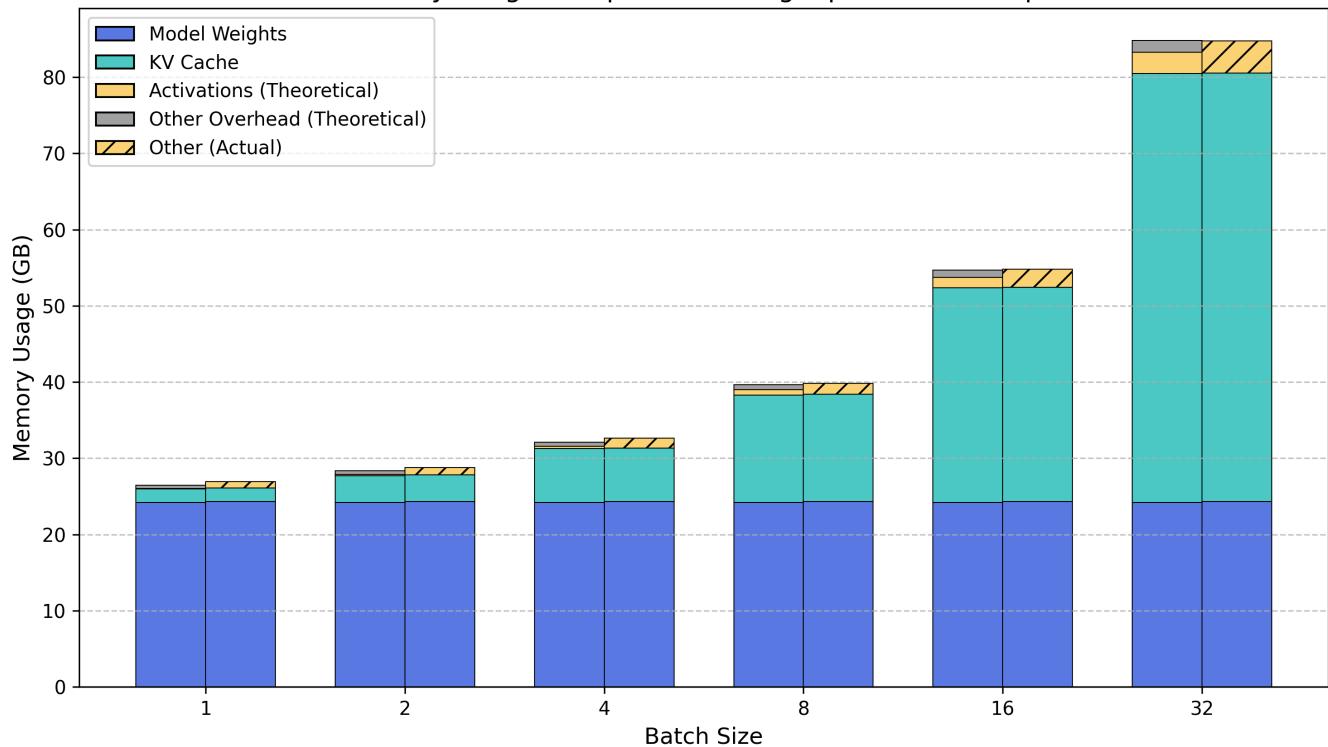
Memory Usage Comparison - Short Input / Short Output



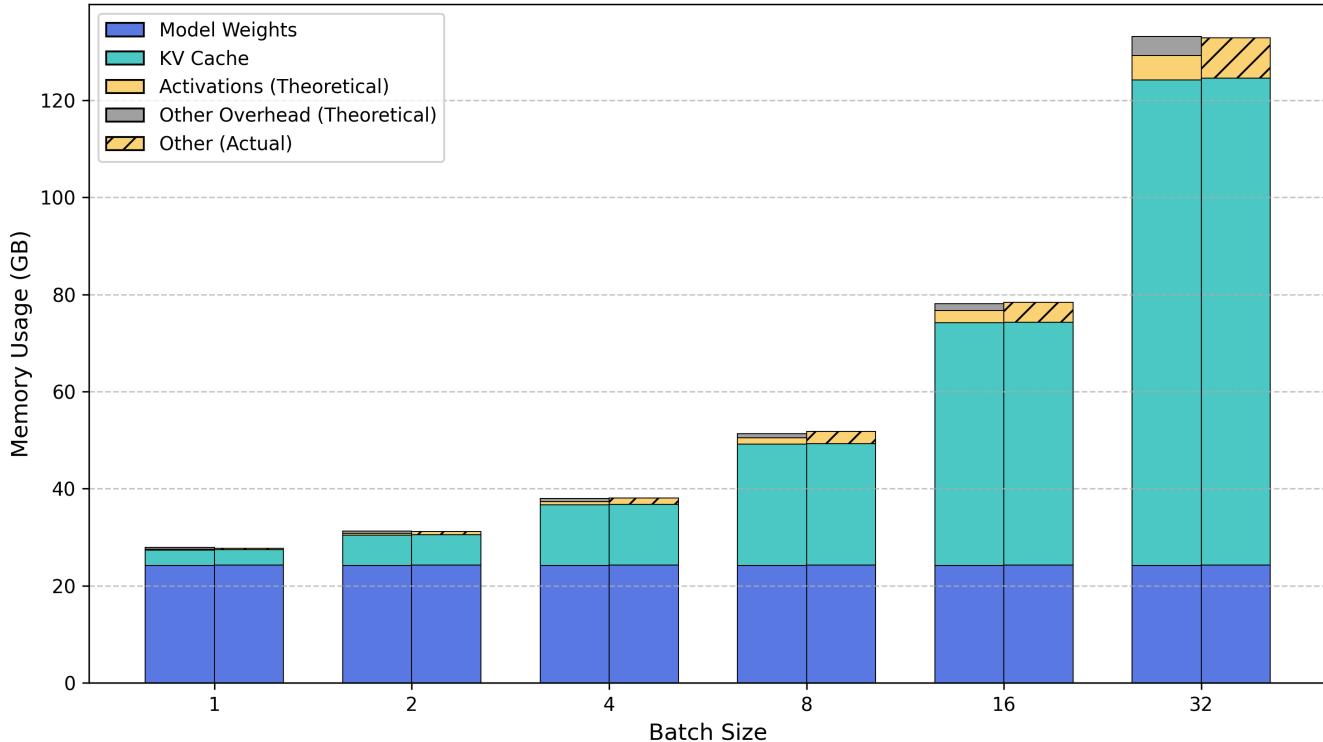
Memory Usage Comparison - Short Input / Long Output



Memory Usage Comparison - Long Input / Short Output



Memory Usage Comparison - Long Input / Long Output



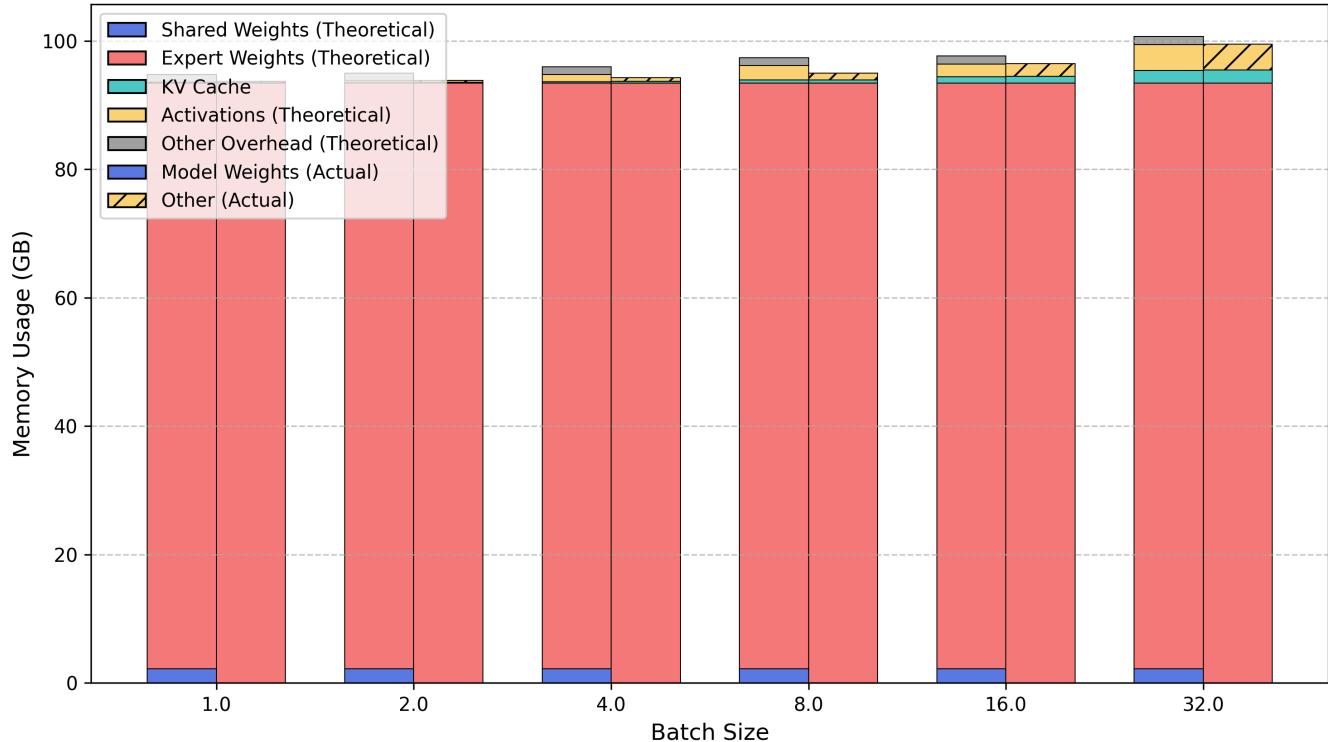
MoeBench

在 MoE 模型的显存测试框架中，我们针对稀疏激活特性构建了差异化的显存估算模型。与 Dense 模型不同，MoE 模型的参数量被解构为共享参数与专家参数两部分：共享参数包含嵌入层、注意力机制及路由门控网络，其显存占用由隐藏层维度与词汇表的乘积（嵌入层）叠加注意力参数构成；专家参数则基于FFN层的专家网络参数量乘以专家总数，采用分设备存储策略时，实际算力需求仅需计算活跃专家参数。

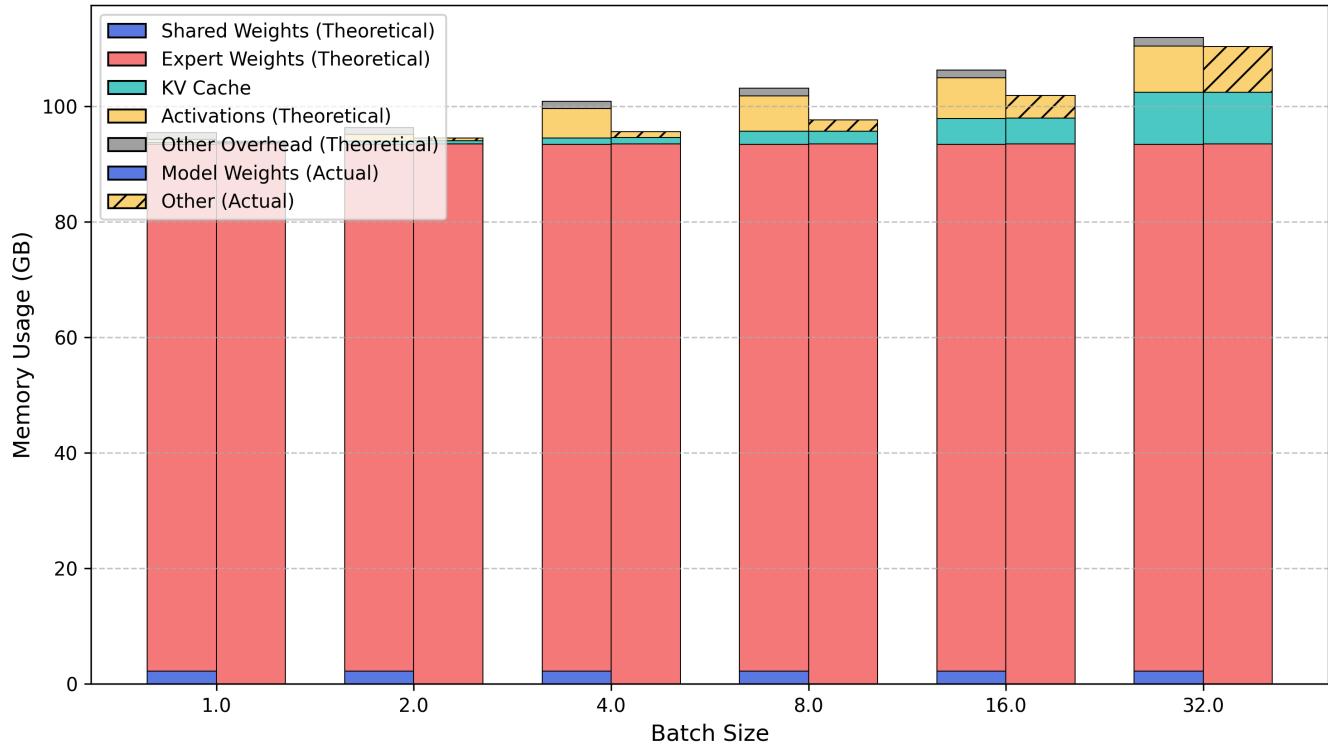
激活值显存计算引入双因子机制，共享激活部分沿用与 Dense 模型相同的 `MOE_SHARED_ACTIVATION_FACTOR`，专家激活部分则采用 `MOE_EXPERT_ACTIVATION_FACTOR`，两者分别乘以序列长度、批处理规模及中间层维度。由于 MoE 模型在推理时仅激活部分专家网络，实测阶段通过遍历模型输出的 `past_key_values` 张量，精确捕获各专家层的显存波动，特别监测专家切换时的显存增量。

结果分析模块新增共享权重/专家权重的显存占比对比曲线，揭示不同批处理规模下模型结构的显存分布规律。当处理长序列任务时，专家激活显存的增长速度显著高于共享部分，这种现象为专家负载均衡策略的优化提供了数据支持。测试报告同时记录路由决策耗时，帮助开发者辨识显存消耗与计算效率的平衡点。

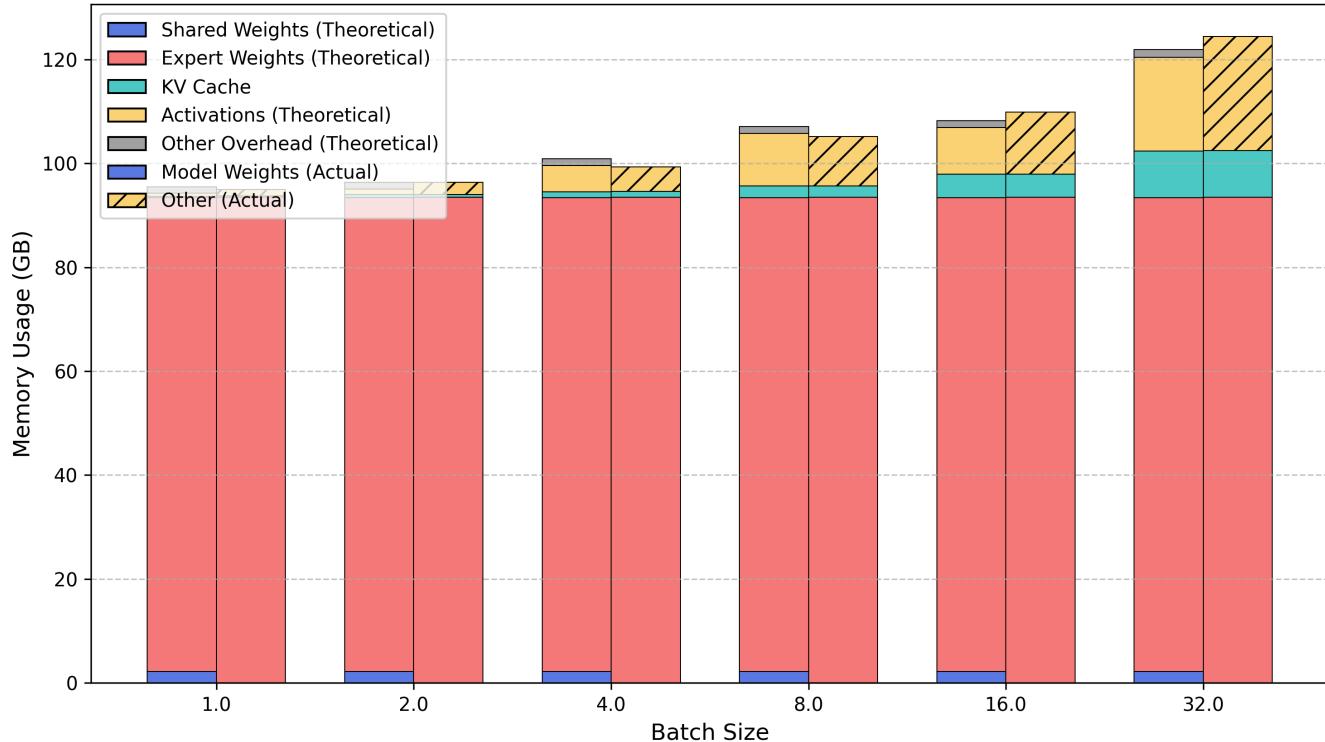
Memory Usage Comparison - Short Input / Short Output



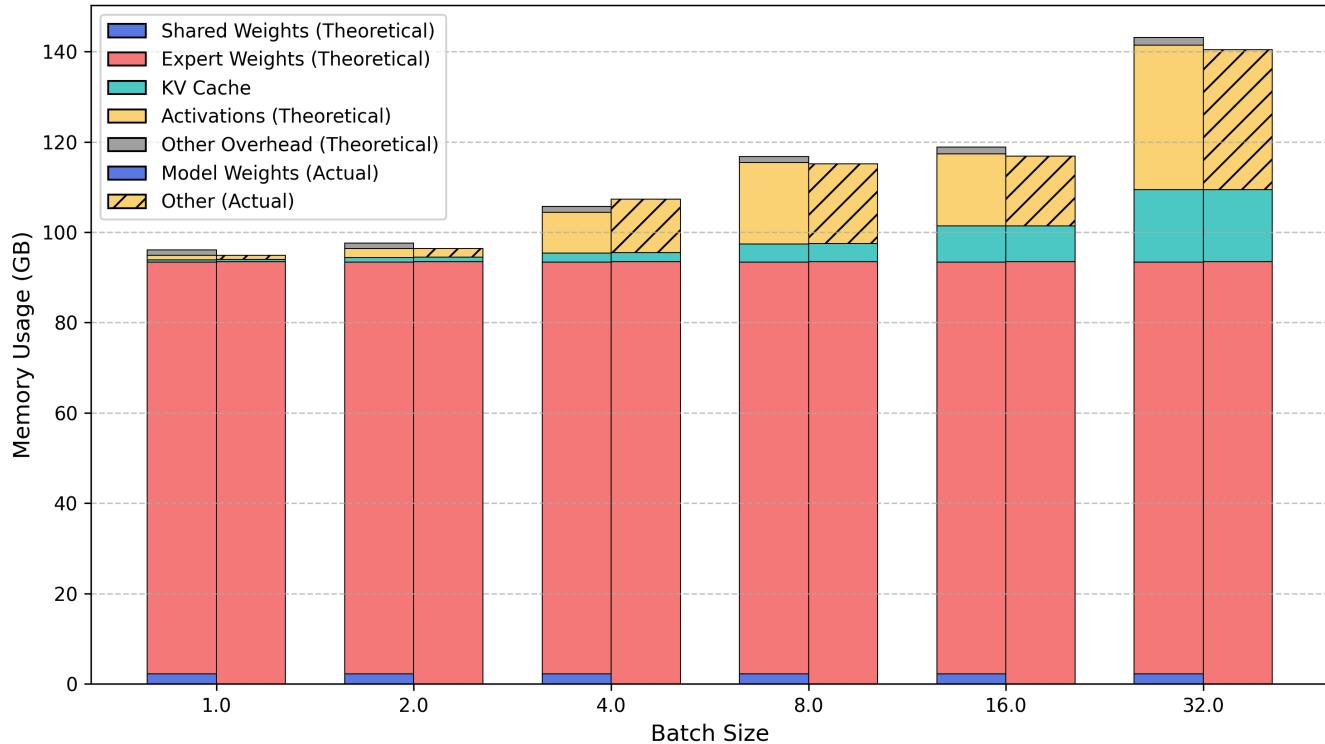
Memory Usage Comparison - Short Input / Long Output



Memory Usage Comparison - Long Input / Short Output



Memory Usage Comparison - Long Input / Long Output



MultimodalBench

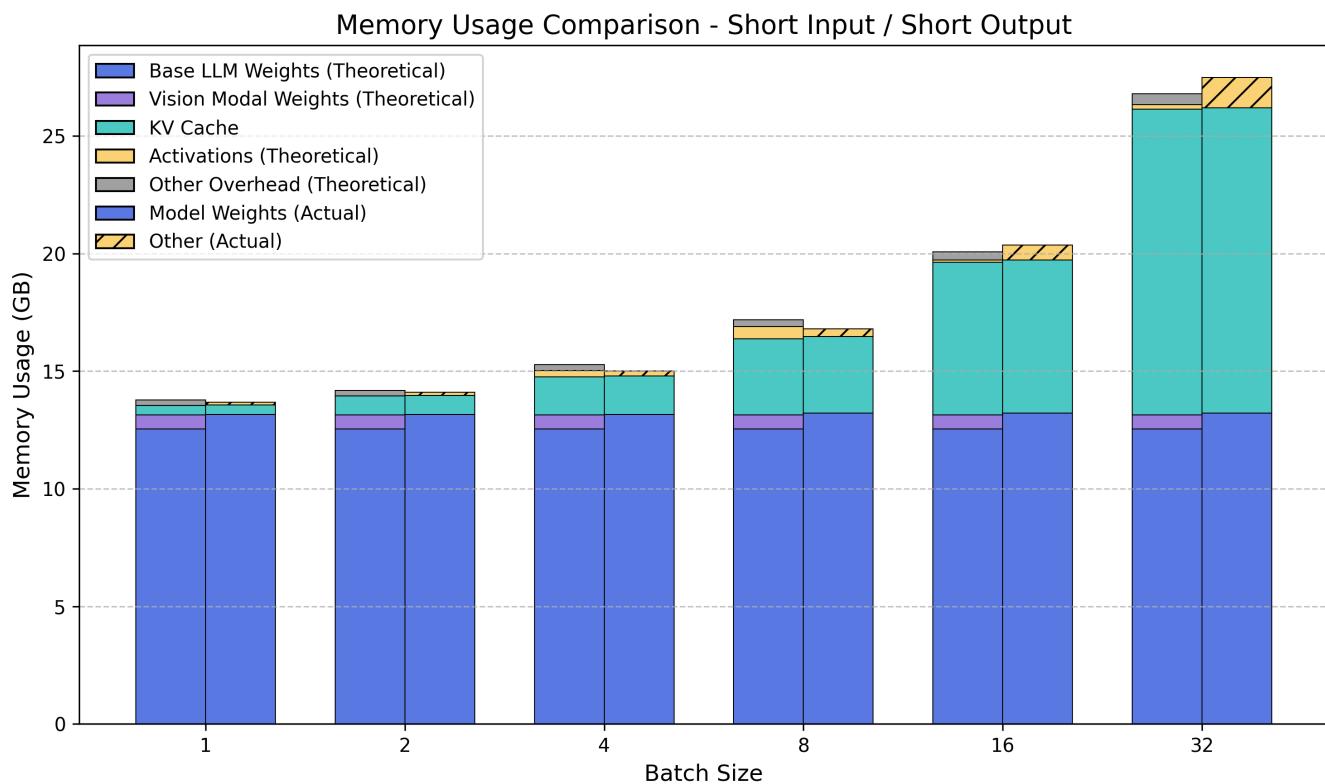
在多模态模型测试中，显存计算采用了实测与理论估算相结合的方法，通过动态测量和参数推导全面评估显存需求。测试脚本首先加载多模态模型，并解析模型配置文件中的关键参数，包括隐藏层维度、注意力头数、图像处理尺寸等配置信息。模型权重通过设备映射分配到多个 GPU 上，同时根据图像 Patch 尺寸动态计算视觉模块的输入序列长度，为后续显存分析奠定基础。

在显存实测环节，测试程序通过 CUDA 接口实时追踪显存状态。每次推理前重置所有 GPU 的峰值显存统计，随后生成模拟输入序列并执行模型推理。在推理过程中，系统精确记录模型权重加载的初始显存占用、KV 缓存的动态增长以及激活值的瞬时消耗，最终通过峰值显存统计接口获取实际显存占用量。针对多模态特性，测试特别处理了视觉模块的图像输入，通过处理器生成符合模型要求的图像张量，确保视觉与文本模态的联合推理符合真实场景。

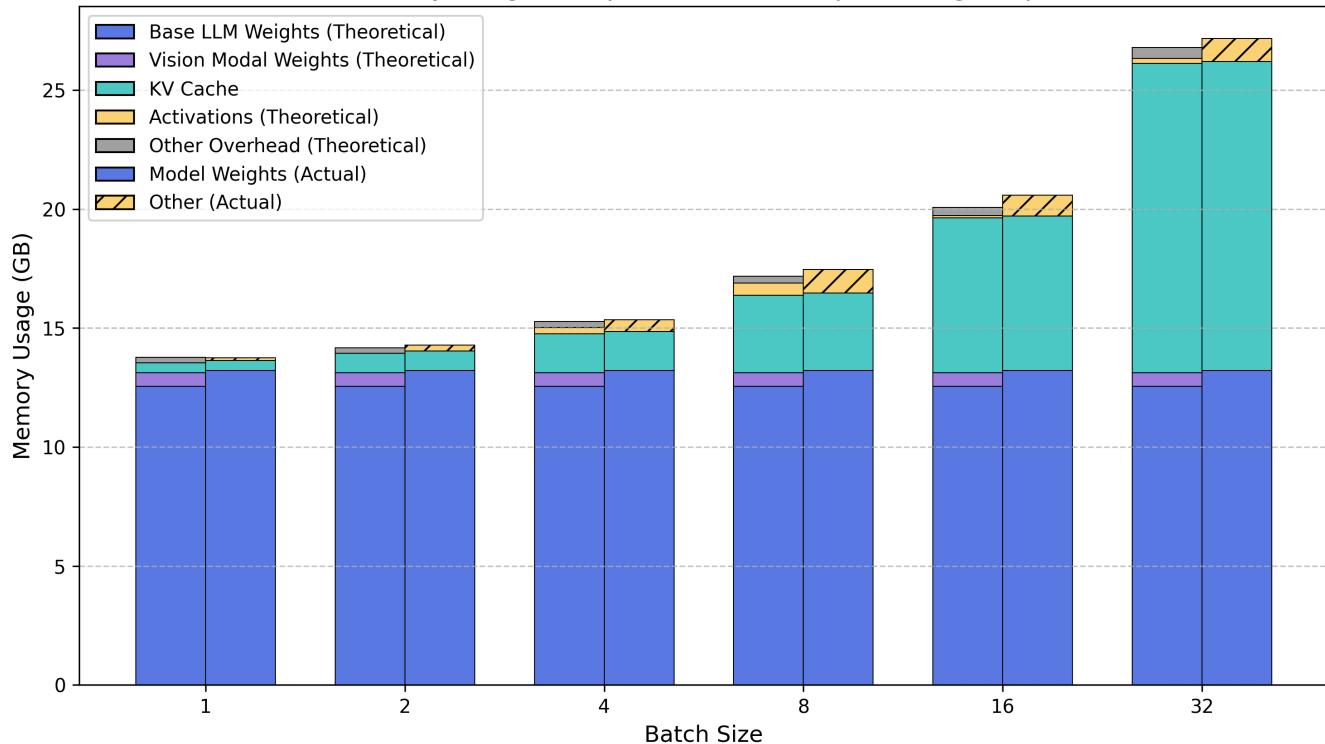
理论计算部分构建了分模块的显存估算模型。模型权重显存通过参数量与量化精度计算，其中基础语言模型、视觉编码器的参数量分别从配置文件动态提取。KV 缓存显存则基于批处理大小、总序列长度（文本 Token 数与图像 Patch 数的加权求和）、注意力层数及头维度进行矩阵式推导，量化精度单独设定。激活值显存通过引入模态相关因子动态调整，准确反映不同模块的计算强度。理论模型还预留了额外开销比例，以覆盖框架层和通信的隐性成本。

测试同样覆盖了典型推理场景的显存压力测试，包括短/长文本输入与生成长度的四种组合，以及多批次规模的横向对比。每次测试后，系统将实测峰值显存、理论分解值（权重 / KV 缓存 / 激活值 / 其他）及推理吞吐量写入结构化 CSV 报告。这种双轨验证机制既提供了即时的显存监控数据，又通过理论模型揭示显存瓶颈的构成因素，为模型部署时的资源预判提供可解释的量化依据。

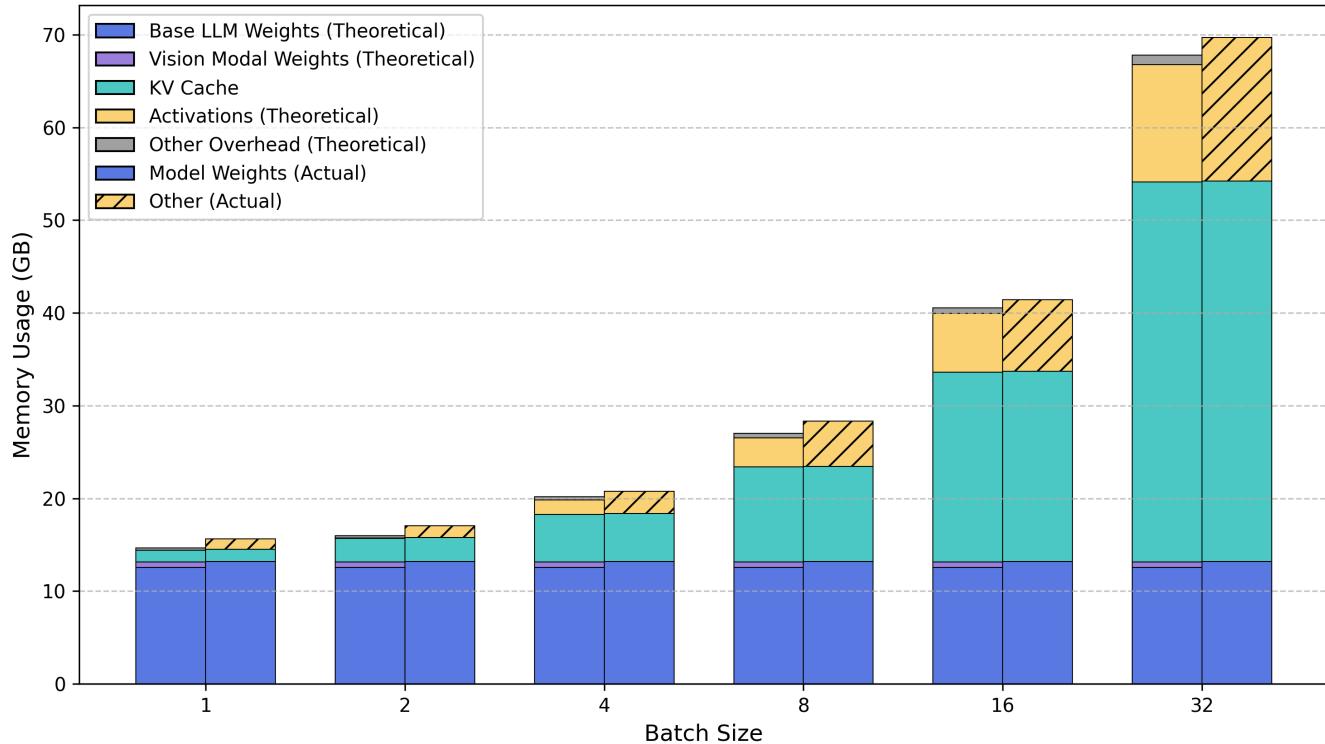
下面是我们对 Mixtral 8x7B (MoE) 模型和 llava-1.5 (Multimodal) 模型的测试结果展示：

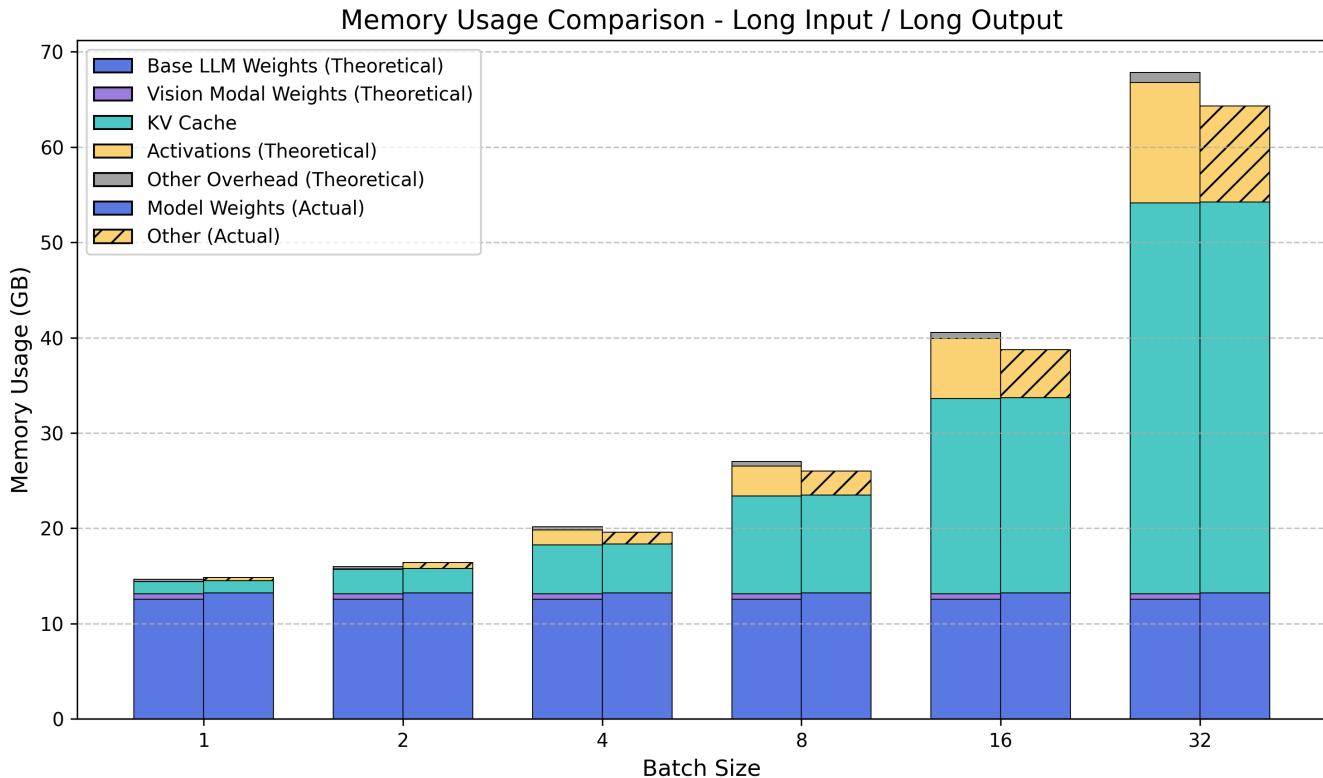


Memory Usage Comparison - Short Input / Long Output



Memory Usage Comparison - Long Input / Short Output





Perf-Test

算力测试部分的实现通过构建通用基准测试框架，评估硬件在不同计算类型和数据类型下的性能表现。测试框架以矩阵乘法 (GEMM) 和层归一化 (LayerNorm) 为代表操作，分别衡量计算密集型任务的理论算力和访存密集型任务的有效带宽。整体设计兼顾测试结果的稳定性和测量精度，采用预热迭代消除初始化干扰，利用 CUDA 事件实现纳秒级异步计时，并通过多次测试取均值降低偶然误差。

测试流程首先通过 `prepare_func` 函数生成特定维度的随机张量，确保每次测试的输入数据具备一致性。预热阶段执行 20 次迭代操作，使 GPU 达到稳定工作状态并完成内核编译。正式测试阶段通过 CUDA 事件精确记录 100 次迭代的总耗时，计算单次操作平均时间时同步等待所有计算核心完成任务，避免流水线并行带来的计时偏差。

性能计算模块根据算子特征采用差异化的评估策略。GEMM 测试基于矩阵维度参数，通过 $2 \times M \times N \times K$ 公式估算浮点运算量，最终输出 TFLOPS 指标反映计算单元的理论峰值利用率。LayerNorm 测试则根据张量元素数量和数据精度计算内存读写总量，输出有效带宽指标反映存储系统的数据传输效率。错误处理机制实时捕获内存溢出、数据类型不兼容等异常，确保测试流程的鲁棒性。

我们的测试中，NVIDIA A100-40G PCIe 的测试结果如下：

| Benchmark Summary: | | |
|--------------------|--------------------|------------------|
| Precision | GEMM (TFLOPS/TOPS) | LayerNorm (GB/s) |
| bf16 | 248.65 | 1078.79 |
| fp16 | 238.68 | 1080.94 |
| fp32 | 16.05 | 970.53 |
| fp8_e4m3 | N/A or Skipped | N/A |

| | 理论算力 | 实测算力 | 利用率 |
|------|------------|---------------|--------|
| FP16 | 312 TFLOPS | 248.65 TFLOPS | 79.70% |
| BF16 | 312 TFLOPS | 138.68 TFLOPS | 76.50% |
| FP8 | N/A | N/A | N/A |

结果显示 NVIDIA A100-40G PCIe 实际的利用率在 80% 作用。因此我们可以在网页中设置计算卡的利用率为 0.8：

The screenshot shows a user interface for configuring GPU resources. At the top, there are two tabs: '推荐配置' (Recommended Configuration) and '自定义配置' (Custom Configuration). Below these, under the heading '推荐硬件' (Recommended Hardware), is a section labeled '异构资源利用率' (Heterogeneous Resource Utilization) with a value of '0.8'. A large red arrow points to this '0.8' value. To the right of the value is a question mark icon. Below this section is a table showing recommended hardware configurations:

| 厂商 | 型号 | 架构 | 显存 | 预估卡数 |
|--------|-----------|-----|-------|------|
| NVIDIA | A100 PCIe | GPU | 80 GB | 17 |
| NVIDIA | H100 PCIe | GPU | 80 GB | 17 |

SLO-test

SLO (Service Level Objective) 是对系统服务质量的量化承诺指标，在人工智能推理场景中主要用于衡量系统响应能力与服务质量。

下面是对其中涉及到的一些指标的解释：

| 指标名称 | 英文全称 | 定义说明 |
|------|-----------------------|---|
| TTFT | Time to First Token | 从用户提交请求到接收到第一个输出 token 的时间间隔，反映系统响应及时性 |
| TPOT | Time Per Output Token | 平均每个输出 token 的生成耗时，衡量大模型持续输出效率 |
| 吞吐量 | Throughput | 单位时间内系统处理的 token 总量 (tokens/s)，体现系统整体处理能力 |

在测试中，我们将 DeepSeek 671B 满血版作为被测模型，通过项目中的自定义硬件方式，指定了某国产加速卡的性能配置 (FP16 算力 1123TFLOPS, 显存 96GB)，并且声明了我们测试的 SLO (TTFT 500 ms, TPOT 50 ms)。

通过我们的 AICompass 项目，我们得出如果要部署项目，需要使用 16 卡的配置。经过 vilm benchmark 的测试，最终得出结论：实际部署的性能能够达到用户目标吞吐量。

- 短输入短输出：

| 并发量 | TTFT(ms) | TPOT(ms) | 吞吐量 (tokens/s) |
|-----|----------|----------|----------------|
| 1 | 56.03 | 20.72 | 47.62 |
| 8 | 156.04 | 32.74 | 237.16 |
| 16 | 221.9 | 37.8 | 407.43 |
| 32 | 349.12 | 45.07 | 673.69 |
| 64 | 524.92 | 56.04 | 1068.52 |

- 短输入长输出：

| 并发量 | TTFT(ms) | TPOT(ms) | 吞吐量 (tokens/s) |
|-----|----------|----------|----------------|
| 1 | 55.9 | 20.99 | 47.61 |
| 8 | 155.8 | 33.73 | 236.72 |
| 16 | 223.13 | 40.32 | 395.93 |
| 32 | 347.34 | 49.04 | 650.52 |
| 64 | 1982.07 | 63.97 | 848.73 |

- 长输入短输出：

| 并发量 | TTFT(ms) | TPOT(ms) | 吞吐量 (tokens/s) |
|-----|----------|----------|----------------|
| 1 | 193.96 | 20.63 | 45.49 |
| 2 | 288.14 | 22.97 | 188.57 |
| 4 | 483.73 | 26.71 | 277.94 |
| 8 | 895.98 | 35.62 | 384.58 |
| 16 | 1455.39 | 46.35 | 352.26 |

- 长输入长输出：

| 并发量 | TTFT(ms) | TPOT(ms) | 吞吐量 (tokens/s) |
|-----|----------|----------|----------------|
| 1 | 195.19 | 20.04 | 49.68 |
| 2 | 288.84 | 22.56 | 234.43 |
| 4 | 472.61 | 26.01 | 388.75 |
| 8 | 895.77 | 33.7 | 462.8 |

| 并发量 | TTFT(ms) | TPOT(ms) | 吞吐量 (tokens/s) |
|-----|----------|----------|----------------|
| 16 | 1456.85 | 40.45 | 483.83 |

参考文献

- Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.
- Han K, Xiao A, Wu E, et al. Transformer in transformer[J]. Advances in neural information processing systems, 2021, 34: 15908-15919.
- Silva L, Barbosa L. Improving dense retrieval models with LLM augmented data for dataset search[J]. Knowledge-based systems, 2024, 294: 111740.
- Wang S, Chen Z, Li B, et al. Scaling laws across model architectures: A comparative analysis of dense and MoE models in large language models[J]. arXiv preprint arXiv:2410.05661, 2024.
- Du X, Gunter T, Kong X, et al. Revisiting moe and dense speed-accuracy comparisons for lilm training[J]. arXiv preprint arXiv:2405.15052, 2024.
- Shen L, Chen G, Shao R, et al. Mome: Mixture of multimodal experts for generalist multimodal large language models[J]. Advances in neural information processing systems, 2024, 37: 42048-42070.
- Liang W, Yu L, Luo L, et al. Mixture-of-transformers: A sparse and scalable architecture for multi-modal foundation models[J]. arXiv preprint arXiv:2411.04996, 2024.
- Li Y, Jiang S, Hu B, et al. Uni-moe: Scaling unified multimodal llms with mixture of experts[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2025.
- Zacarias F V, Palli K, Vazhkudai S, et al. Analyzing LLM performance: The impact of high-bandwidth memory on model inference[J].
- Na S, Jeong G, Ahn B H, et al. Understanding performance implications of lilm inference on cpus[C]//2024 IEEE International Symposium on Workload Characterization (IISWC). IEEE, 2024: 169-180.
- Hasan S, Basak S. Open-source AI-powered optimization in scalene: Advancing python performance profiling with DeepSeek-R1 and LLaMA 3.2[J]. arXiv preprint arXiv:2502.10299, 2025.
- Gao T, Jin J, Ke Z T, et al. A comparison of deepseek and other LLMs[J]. arXiv preprint arXiv:2502.03688, 2025.
- Zhao K, Liu Z, Lei X, et al. Quantifying the Capability Boundary of DeepSeek Models: An Application-Driven Performance Analysis[J]. arXiv preprint arXiv:2502.11164, 2025.