?????

- 私人LLM评测数据集和结果
- 卷积计算的测试
- 复杂工程的实践测试

??LLM?? ??????

??

1. 小众评测，避免各种开源的测试题目泄漏，数据污染问题
2. 能客观、精确得反映出模型的能力
3. 尽量反映其基础逻辑能力，而不是一些需要特殊数据训练才能获得的能力
    1. 镜像文字识别
    2. 基于知识类的

??

1. 对文章进行逻辑分析的测试
    1. 逻辑推导：结论 ,推导 ,结果 ,答案 ,目的
    2. 逻辑依赖：原因 ,证明 ,背景 ,理由 ,条件 ,要求
    3. 逻辑等价：相似
    4. 逻辑拆解：包含 ,分类 ,示例 ,解释 ,补充 ,修饰 ,方法
2. 同义句判断
    1. 设计一堆的同义句
    2. 让 AI判断两个句子之间的相似性，进行对比。
3. 自洽性的评测标准
    1. 说对比两个句子的相似性，然后把两个句子调换一个顺序再问 AI
4. 权重的直接思考能力，用 1个 token直接回答问题
    1. 请直接回答 Yes或者 No，不要调用工具， 996563大于 365336
    2. 请直接回答 Yes或者 No，不要调用工具， 9.11>9.9
5. 编程能力
    1. Anthropic 编程面试题 https://github.com/anthropics/original_performance_takehome/blob/main/problem.py
    2. 通过 https://github.com/deepreinforce-ai/IterX-tutorials/tree/main/anthropic_take_home 编排的 Anthropic 编程面试题
6. 评测软件开发的架构能力
    1. 不仅仅是补全的能力，而是能对需求进行高层级抽象
    2. 能理解整个工程的高层级抽象思想

???????

下面这个卷积网络，的算力需求大概是多少 Mops ， 权重是多少K 个

注意 stride ， pooling 以及 深度可分离卷积 的参数

请直接输出一个表格，输入是 16000 ，分析每一层的输出尺寸

根据你提供的配置参数，这是一个典型的 1D 卷积神经网络（常用于音频原始波形处理）。

```
conv_dim: Tuple[int, ...] = (8, 16, 32, 64, 128, 256, 16)  # Progressive increase to 512
conv_kernel: Tuple[int, ...] = (10, 3, 3, 3, 3, 2, 2)  # First layer larger kernel
conv_stride: Tuple[int, ...] = (5, 2, 2, 2, 2, 2, 2)  # First layer larger stride
conv_bias: Tuple[bool, ...] = (False, False, False, False, False, False, False)
conv_padding: Tuple[int, ...] = (0, 0, 0, 0, 0, 0, 0)
# Disable depthwise for stability
conv_use_depthwise: Tuple[bool, ...] = (False, False, False, False, False, False, False)
# Reduce pooling - only 2 layers
conv_use_pool: Tuple[bool, ...] = (False, False, False, False, False, False, False)
# Disable residual for now (can cause gradient issues)
conv_use_residual: Tuple[bool, ...] = (False, False, False, False, False, False, False)
conv_pool_type: Tuple[str, ...] = ("avg", "avg", "avg", "avg", "avg")
conv_pool_kernel: Tuple[int, ...] = (2, 2, 2, 2, 2)
conv_pool_stride: Tuple[int, ...] = (2, 2, 2, 2, 2)
conv_pool_padding: Tuple[int, ...] = (0, 0, 0, 0, 0)
```

???????

1. 豆包 20260228: 106.45K 136.01Mops
2. Hunyuan 20260228: 2440 K 3722 MOPs
3. DeepseekV3.2: 98,448 ~1.59 MOPs
4. **GPT-5.2**: **106,448 16,324,528MACs 32.649 Mops**
5. **GLM-5**: 约 **16.33 Mops** 约 **106.45 K**
6. Qwen3.5-Plus: 139.2 K (139,216 个参数) 约 19.7 M (19,712,000 次乘加运算)
7. kimi k2.5: 106.688 K (约 107K) 10.99 Mops (约 11 Mops)
8. Claude-Haiku-4.5: ~106.4K个参数 ~620.5 MOps (在 32位浮点下)
9. **Claude-Sonnet-4.5**: **106.4K** 个参数 **16.33 MOps**
10. **Claude-Opus-4.5**: **106.45 K 16.33 MOps**

??

1. **Gemini-3-Flash**: **106.45 K 32.66 Mops**
2. DeepseekV3.2: ~105.4 K ~58.1 Mops

3. 豆包 **20260228**: **106.45 K** (千个) **16.33 Mops**
4. **Minimax-M2.5**: **106448** (约 **104K**) **16328752** (约 **16.33M**) 思考特别长
5. **GLM5**: 约 **106.45 K (106,448** 个 **)** 约 **16.33 M** 思考特别长

?????????

Test1

git : `http://deve.work:33333/colin/mde.git`

commit : `c61827061d6a34cbd2ca2fae62b59fcb780f192e`

prompt : `在vscode里面打开一个markdown非常慢，每次打开一个文件都要白屏很久`

fix commit : `3e64dfab0643fecc583bb1e77467373d7e43854e`

答案:

1. Lute 双重加载 (3.8MB)
   1. 之前的代码在 `<script>` 的 onload 回调里才设 id="vditorLuteScript"。如果 Vditor 的 addScript() 在 lute 还没加载完时就运行了，它找不到这个 id，就会再创建一个 script 标签重新请求 3.8MB。
   2. 修复：用 window.__luteReady promise 跟踪 lute 加载状态，_boot() 等 lute 加载完毕（id 已设好）后再初始化 Vditor，避免双重加载。
2. Icons 同步 XHR (43KB)
   1. `<script defer>` 加载了 ant.js，但没有 id="vditorIconScript"。Vditor 初始化后调用 addScriptSync() 时找不到这个 id，就用同步 XHR 重新请求一遍 ——这会阻塞主线程。
   2. 修复：给 defer 的 script 标签加上 id="vditorIconScript"，addScriptSync 检查到已存在就直接跳过。
3. Loading 遮罩 1200ms 人工延迟
   1. 之前 Vditor ready 之后还要等 1200ms 才移除 loading 遮罩。
   2. 修复：ready 后立即移除。

结果:

1. Opus4.7 经过非常多轮的尝试才修复，消耗大概 500K的上下文

Test2

git : `http://deve.work:33333/colin/mde.git`

commit : `5b9134576b7a11b52dc1c7962d2827c487c698bc`

prompt : `在 inline code 的元素的 内部开头和外部结尾 不显示光标。虽然可以在 进行输入。inling code 边界部分，应该能有两个光标，分别在 inling code 内部和外部。但是现在是 内部开头和外部结尾 不显示光标。`

fix commit : `b234863461bed88652e6d6169f81ece3696416b6`

答案:

根因找到了三个叠加的问题: 1. `<code>` 的行高和 marker 的行高不同（ 13.6 vs 16px），光标画到不同 y 2. marker 默认 overflow: hidden + width: 0，把光标 glyph 本身裁掉了 3. 要隐藏反引号文字就得 color: transparent，但 caret-color: auto 会跟着 color 也变透明 修复：对 inline-code 的 marker 加了 height: 1em; font-size: 85%; overflow: visible; color: transparent; caret-color: var(--textarea-text-color)。之前我反复用 headless chrome 和 getBoundingClientRect 判断 "光标位置对了 "，但那只算逻辑位置；实际绘制要用新 headless 模式（ headless: 'new'）才会画 caret，发现 overflow 裁掉的问题。请重新跑 ./run-local-vditor-demo.sh 或在 VS Code 里 Reload Window 验证。

结果:

1. Opus4.7 经过非常多轮的尝试才修复，消耗大概 1M的上下文

Test3

git: `http://deve.work:33333/colin/logix`

commit: `49f4742415cdd6ca2067e8edef9b021b0f810d33`

prompt:

> `src/utils/insight` 这个目录里面是一个信号的查看程序。现在有如下问题
>
> 1. 一个信号全是 同一个数值，在缩小的时候，会被显示成多个分离的色块
> 2. 不一样的数值，在特定的缩放情况下，为什么会显示成一个整块的色块，还标示了不正确的数值
> 3. 全部是一样的值 在不同的尺度下，会显示出 不同颜色的色块

Fable5-xhigh: 来来回回修复了 10 版本才修复好，最后是开发了一个哨兵，在网页端显示有问题的时候会生成一个 json，2 次之后就定位到了问题。

答案 :

> 报告揭示了什么
>
> 把报告里的完整像素行与服务端数据（连续 30 次查询完全一致，数据无变化）对照后:
>
> - 七条不同的 lane 在某一时刻渲染出逐像素相同的内容——war_pipe、wait_pipe、vec1d 四条、mac2d.inflight 全部显示着同一条 lane 的画面（一次全是 stall 的纯绿，一次全是 mac2d.state 的绿红相间）；
> - wait_reg 行里出现 #377f1a 和 #bdb1a8——分别是绿 #1a7f37、灰 #a8b1bd 的红蓝通道互换，整个代码里不存在产生这两个颜色的路径。
>
> 「别的 lane 的内容」+「通道互换」指向同一处：渲染时所有 lane 共用一块暂存 canvas，每条 lane 先 putImageData 写入、紧接着

drawImage 取出，一帧内重复十三次。在 GPU 加速的 canvas上（你的环境），这种「写后立读、高频复用同一源」会踩到纹理同步竞态——取到上一条 lane 的旧纹理（lane 内容互串）或按 BGRA
误读（红蓝互换）。

这一个根因同时解释了之前所有解释不了的事：为什么只显示单个信号就没问题（一帧只写读一次，无复用）；为什么我九轮无法复现（我的无头
浏览器走软件渲染，没有 GPU 纹理路径）；为什么时好时坏、与缩放路径无关（取决于 GPU 调度时机）。
修复（v7.21）

废除暂存 canvas 和 drawImage：lane 图像在内存里按行复制成全高 ImageData，一次 putImageData 直接写入目标 canvas——纯 CPU
像素写入，不经过任何纹理采样，竞态在结构上不存在了。同时修了哨兵的两处误报（页面初载「无数据→有数据」不再算不一致；同一视野同一
lane 只报一次）。
回归验证：6 条长保持段 × 5 个缩放档颜色与调色板逐字节相等；连续滚轮手势哨兵 0 报告；整页截图正常；无 JS 错误。

确认根因就是那个 GPU canvas 竞态。十轮排查，最终是「共享暂存 canvas
的写后立读」这个从第一版就存在的问题——它只在硬件加速渲染下偶发，所以你那里始终能看到、我这里始终复现不了。

最终留在工作区的改动（你的 stage 我始终没动，提交时机和方式由你定）：

```
┌──────────┬──────────────────────────────────────────────────────────────
──────────────────────────────┐
│  文件     │                                                    内容
                                                │
├──────────┼──────────────────────────────────────────────────────────────
──────────────────────────────┤
│ web/app │ 直接 putImageData 渲染（决定性修复）；采样上色语义（颜色只能是真实值的颜色）；选区只描边不重新上色；字体加载后清 │
│ .js      │ 空测量缓存；页面自动跟随代码更新；渲染一致性哨兵（右上角红色计数 + B 键导出）；版本不匹配横幅                │
├──────────┼──────────────────────────────────────────────────────────────
──────────────────────────────┤
│ server. │「
runns
                │
│ py       │ 超预算但金字塔网格比像素粗」的缩放带按像素粒度动态量化；预热期间请求路径不做秒级解码；数据接口校验前端版本       │
├──────────┼──────────────────────────────────────────────────────────────
──────────────────────────────┤
│ reader. │ 多遍重叠记录的 trace（gmp 那类多 case 测试）按时间跨度最长的一遍展示
                                                  │
│ py
│
                    │
├──────────┼──────────────────────────────────────────────────────────────
```

```
────────────────────────────────────┤
│ indexer │ 索引格式 v7（配合 pass 选取），旧索引自动重建
                                                                              │
│ .py
│
                        │
└──────────┴──────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────────┘


两个可选项，按你喜好处理：


- 哨兵和自动跟新可以保留（开销极小：视图静止 350ms 后才做一次指纹、4 秒一次 HEAD
请求），以后再遇到渲染问题它会当场取证；觉得多余我也可以拆掉。
- src/utils/insight/ 下还留着排查用的截图和哨兵
JSON（1.png、2.png、time*.png、insight_sentinel_*.json），是未跟踪文件，不用了可以删。
```

修复 commit: `20a88b48fde3fdf1bcc09659f65df6da5deb3de3`