终端大模型推理优化-ph2

1. 开发板介绍

润和HH-SCDAYU200开发套件，Rockchip RK3568(arm架构64bit)，支持蓝牙、Wi-Fi、音频、视频和摄像头丰富的扩展接口，支持多种视频输入输出接口。

2. 任务介绍

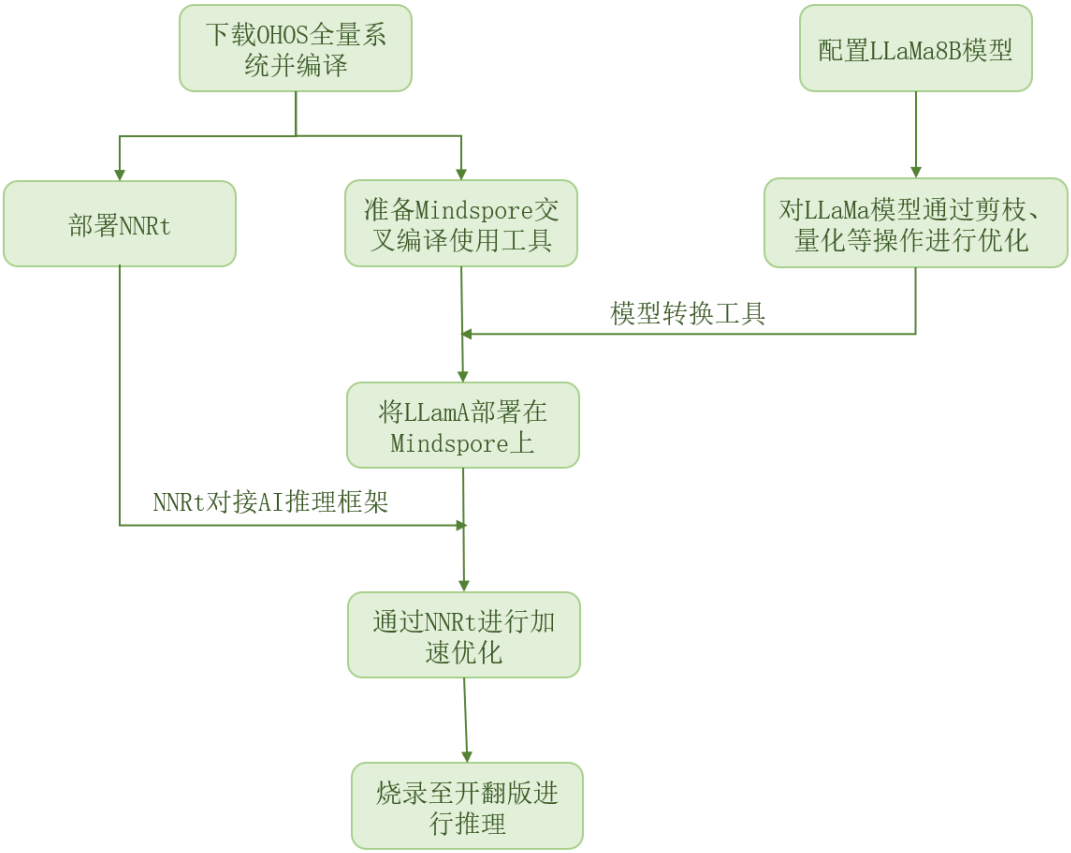
终端大模型推理性能优化

> 在本赛题中，我们将选用LLaMA作为Workload，LLaMA的版本和参数量不做要求。模型可在Huggingface上进行下载。

- 优势：提高隐私安全性、降低云端通信时延等
- 挑战：终端设备通常具有有限的计算资源和内存容量
- 优化技术
    1. 大模型压缩技术[1,2]：量化、剪枝、知识蒸馏和低秩分解等
    2. 大模型内存管理优化技术[3]：PagedAttention等
    3. 简单实现(cpu)

最优解应该是使用华为的NPU进行加速，但需要较多的上层支持（mindspore/pytorch 及模型源码）。

以下是初始规划，但受限于OH社区生态较为贫瘠，可参考方案较少，最终没能实现。



退而求其次，采用了避开OH系统与其他依赖难以调和的问题，直接使用llama.cpp编译得到二进制可执行文件，进而直接在开发板运行。

3. 实验设置

3.1 OH鸿蒙系统

以下并没有使用鸿蒙系统的特异性功能，而是将其视作一个linux系统。鸿蒙系统的安装请参考该系列的其他实验文档。

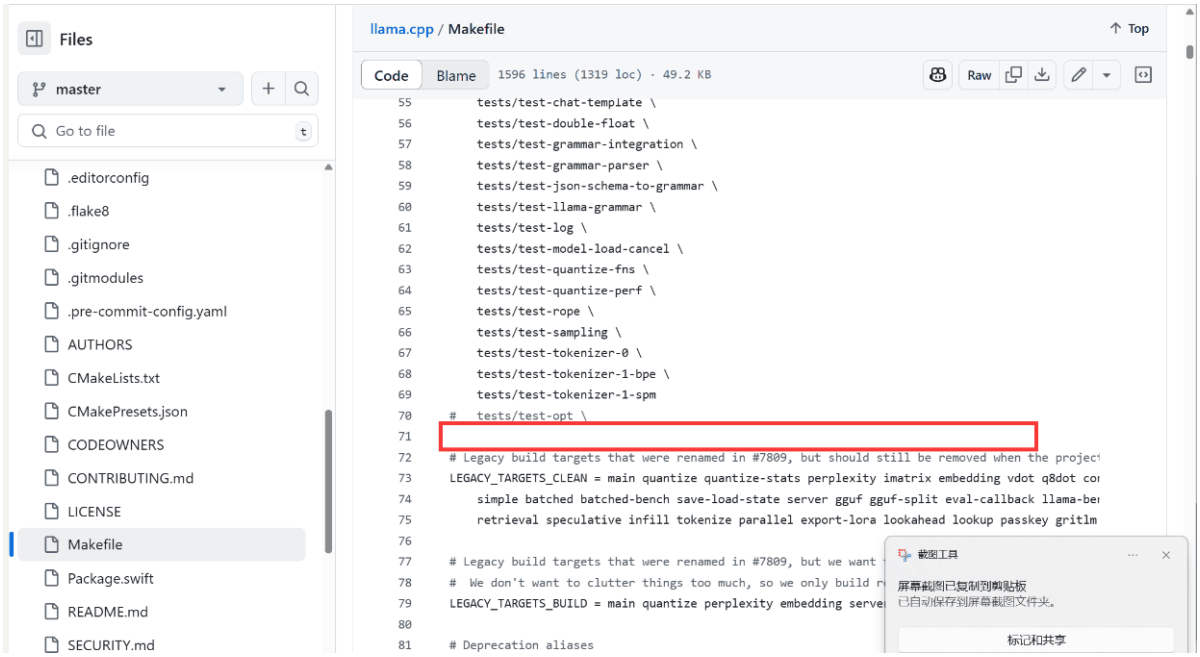
3.2 llama.cpp 的静态编译

```
git clone https://github.com/ggerganov/llama.cpp
cd llama.cpp
```

> 开发板为鸿蒙环境，安装cmake以及依赖较为复杂。所以我们选择将llama.cpp在其他arm架构的平台先行编译（cmakelist和makefile都较为复杂，同样不建议交叉编译；此处我们使用阿里云平台arm服务器），然后直接移植可执行文件。

```
#静态编译
CFLAGS+=-static
CXXFLAGS+=-static
LDFLAGS+=-static
MK_CFLAGS+=-static
MK_CXXFLAGS+=-static
MK_LDFLAGS+=-static
```

配置静态编译，将以上几行插入到：



随后在阿里云arm服务器上执行：

```
cmake -B build
cmake --build build --config Release
```

> 参考https://github.com/ggml-org/llama.cpp/blob/master/docs/build.md

3.3 迁移llama.cpp的二进制文件

运行完成后，在build/bin中复制llama-cli，即生成的可执行二进制文件，可以直接在arm结构上运行。此二进制文件从远程服务器经由本机传递至开发板。

3.4 迁移模型参数

需要提前将对应的gguf文件传送到开发板。

> gguf是一个专门为了llama cpp适配的模型参数格式

我们可以直接下载以gguf结尾的文件，然后传递到开发板上。

以下是我们这次使用的模型，模型较大，传递时间比较长。

模型	大小
Meta-Llama-3.1-8B-Instruct-Q2_K-GGUF · 模型库	3G
Meta-Llama-3.1-8B-Instruct-Q8_0-GGUF · 模型库	8G

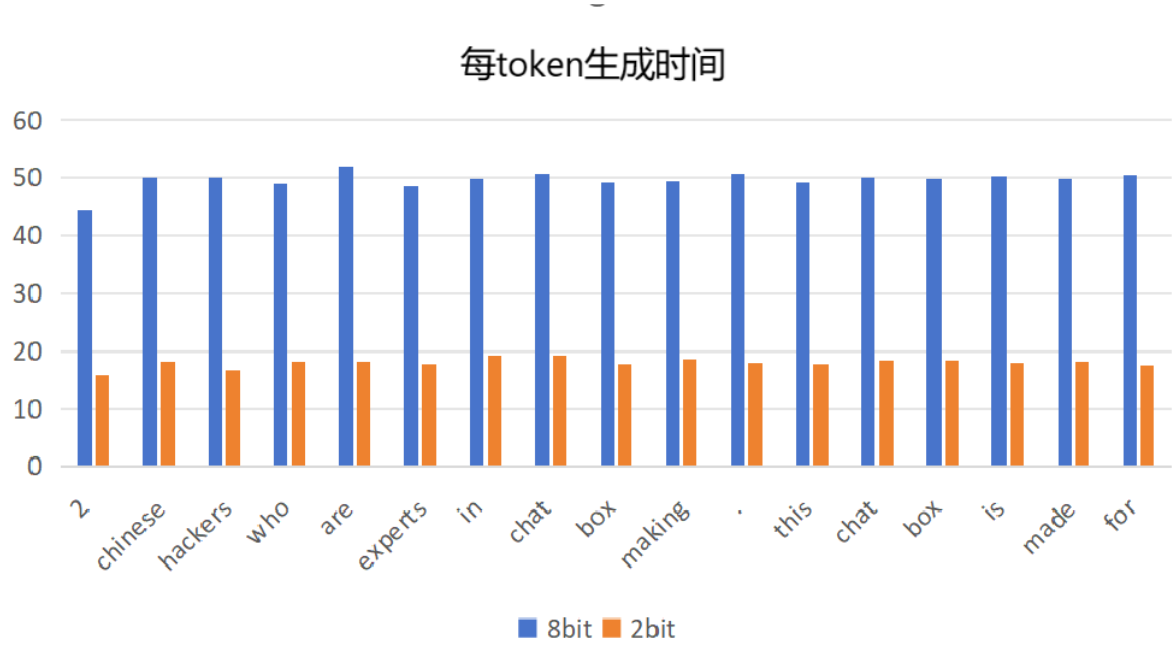
4.模型时间测试

采取续写模式测试运行时间

```
$ llama-cli -m model.gguf -p "I believe the meaning of life is" -n 128 -no-cnv
# I believe the meaning of life is to find your own truth and to live in accordance with it. For me, this means being true to myself and following my passions, even if they don't align with societal expectations. I think that's what I love about yoga – it's not just a physical practice, but a spiritual one too. It's about connecting with yourself, listening to your inner voice, and honoring your own unique journey.
```

> 参数`-no-cnv`，非对话模式，即设置续写模式。

同时记录每一个token的生成时间，得到如下数据：

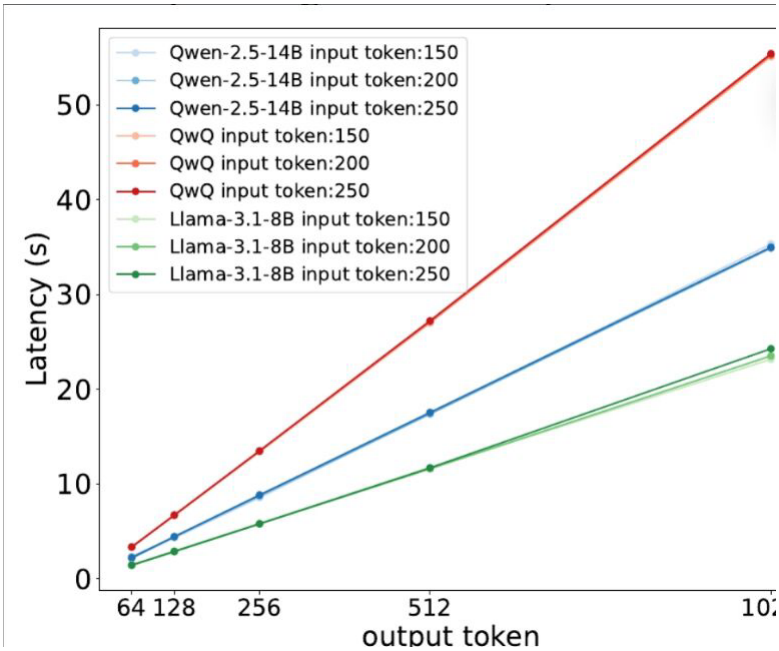


token_id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
8bit	44.35	50.06	50.04	48.90	51.95	48.44	49.71	50.57	49.06	49.38	50.57	49.53	49.53	49.85	50.19	49.73	50.54	50.12	49.49
2bit	12.98	15.90	18.25	16.78	18.17	18.19	17.76	19.22	19.16	17.86	18.62	17.92	18.30	18.31	17.95	18.20	17.62	18.18	19.66

整理后的数据如下，两者时间差距约为2.73倍。

模型	时间
Meta-Llama-3.1-8B-Instruct-Q2_K-GGUF	18.18s
Meta-Llama-3.1-8B-Instruct-Q8_0-GGUF	49.73s

> 扩展内容
>
> （论文研究表明，在输入输出较短时，新单个token生成的时间并不会随着序列的增长有太显著的变化，这是由于kv-cache的存在导致的新生成一个token的运算量相对稳定）



5.模型正确率测试

5.1 实验设置

我们选用了gsm8k数据集(小学数学题)来测试效果。并采取了开源的评分代码来辅助我们进行打分。

> 评分：QwenLM/Qwen2.5-Math: A series of math-specific large language models of our Qwen2 series.

由于开发板运行速度过慢，我们将同样的模型部署到阿里云服务器，并开启网络端口。QWen-math评测代码调用该网络服务，将问题通过网络传递到服务器，获取服务器的回答，抓取答案并评分。

5.2 无延迟限制的推理

直接提问模型数学问题，prompt如下

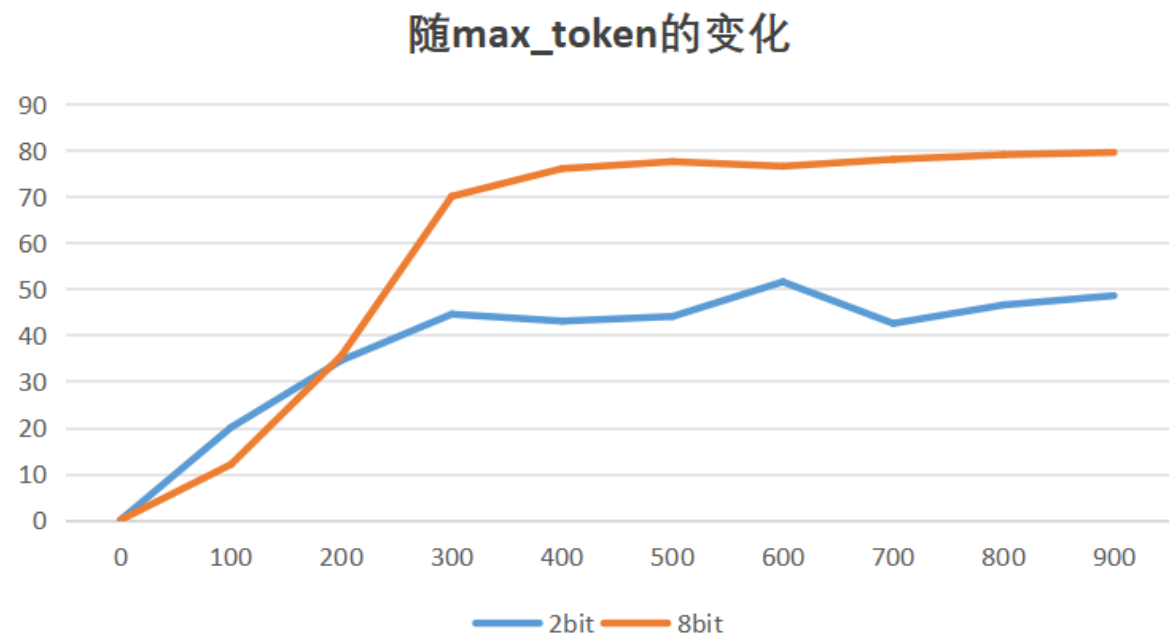
```
<|im_start|>system
Follow this pattern to solve the problem:
Provide a revised answer through step - by - step reasoning .

Conclude with your final answer within \boxed{}.
<|im_start|>user
（实际的数学问题）
<|im_end|>
<|im_start|>assistant
```

模型	正确率
Meta-Llama-3.1-8B-Instruct-Q2_K-GGUF	51.6%
Meta-Llama-3.1-8B-Instruct-Q8_0-GGUF	70.9%
Meta-Llama-3.1-8B-Instruct-full	85.6%

5.3 budget-aware推理

在端侧推理时，我们往往需要限制latency，要能够迅速的完成推理。限制推理最简单的方式就是设置max_token,限制输出答案的长度。我们首先对于这两个模型在不同max_token下的表现做了对比。



我们观察到在token较小时两者效果都受到了比较大的影响。由于Qwen-Math在评测是只关注最后结果，且必须要求\box{}进行包裹答案，所以直接对答案截取可能会错失最后答案，为此我们做了两个调整

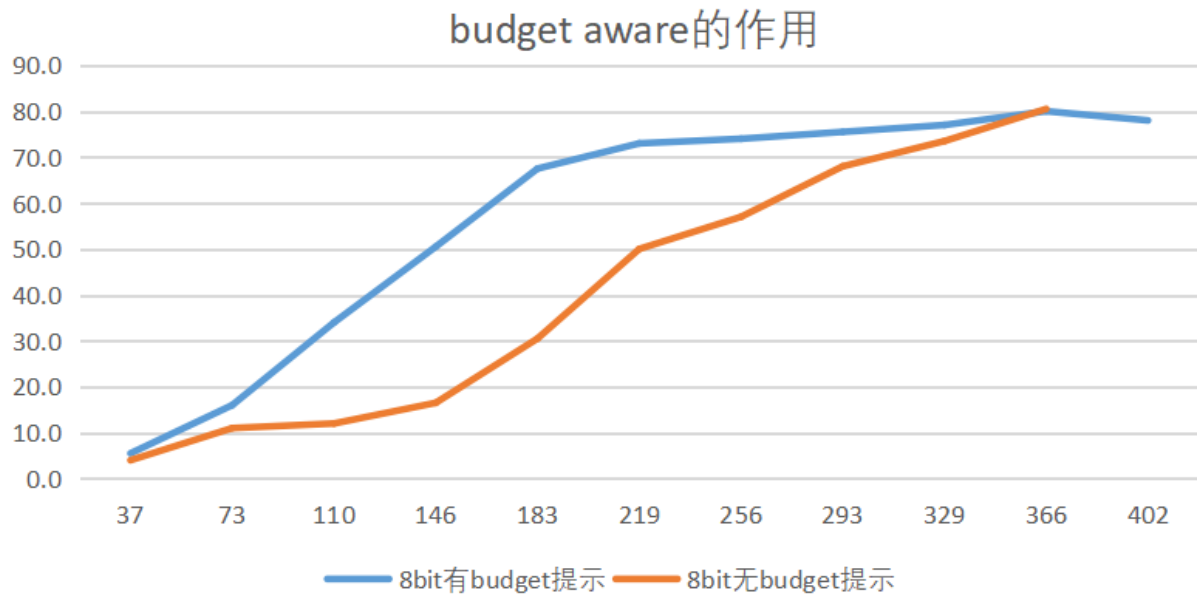
1. 修改prompt为：回答并验证（aav）。这个提示信息能让模型快速给出一个初始答案，然后通过迭代的方式进行验证和修正，这种方式可能对于输出截断问题更具鲁棒性。

> **Quick Answer**: Give an initial answer via intuition or quick calculation.
> **Verification**: Provide a revised answer through step - by - step reasoning and correct any mistakes.
> Conclude with your final answer within \boxed{}.

2. prompt中告诉模型我们的budget限制。考虑到模型经常会超出budget，我们在模型调用时告诉的budget是max_token的一半。

```
prompt=prompt+f"""You MUST respond in **UNDER {int(int(budget)*0.5)} tokens**"""
```

修改后，在prompt受限情况下，模型的效果显著提升了。

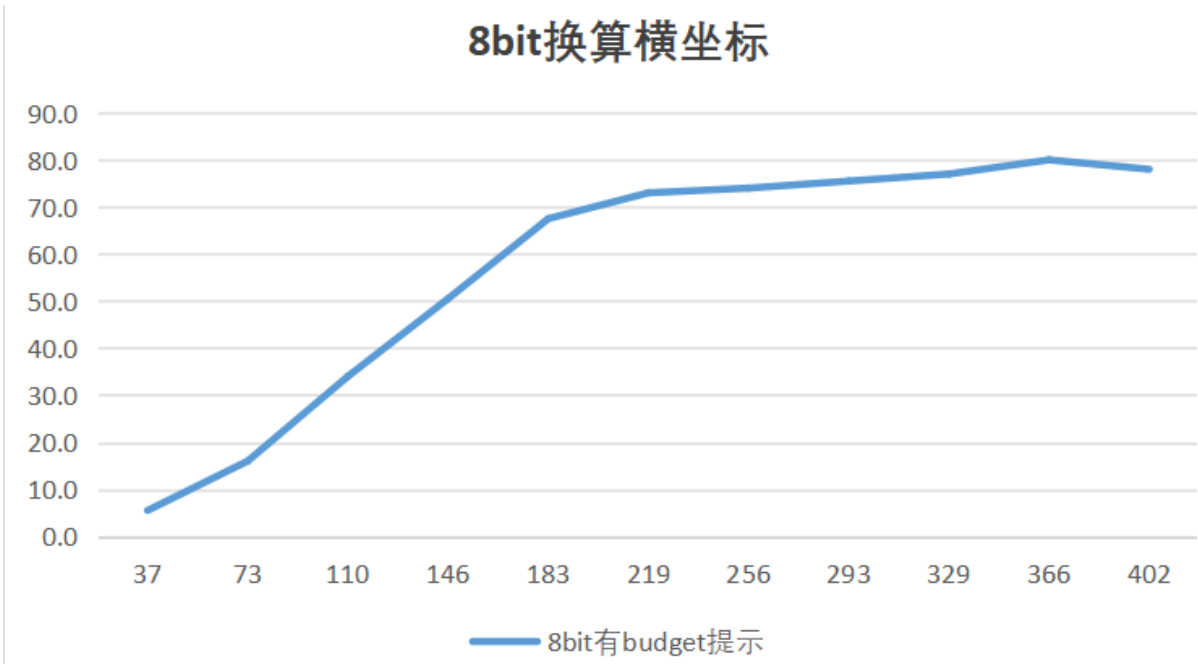


5.4 latency限制的推理

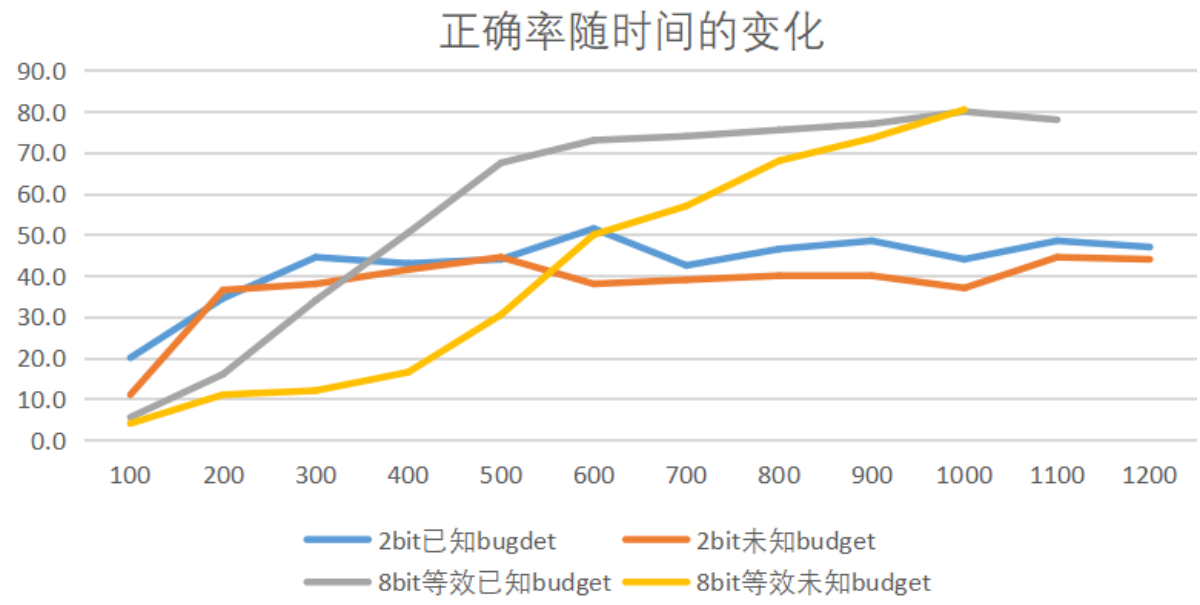
实际生活中。我们关心的往往不是“token相同时，答案的质量”，而是“耗时相同时，答案的质量”

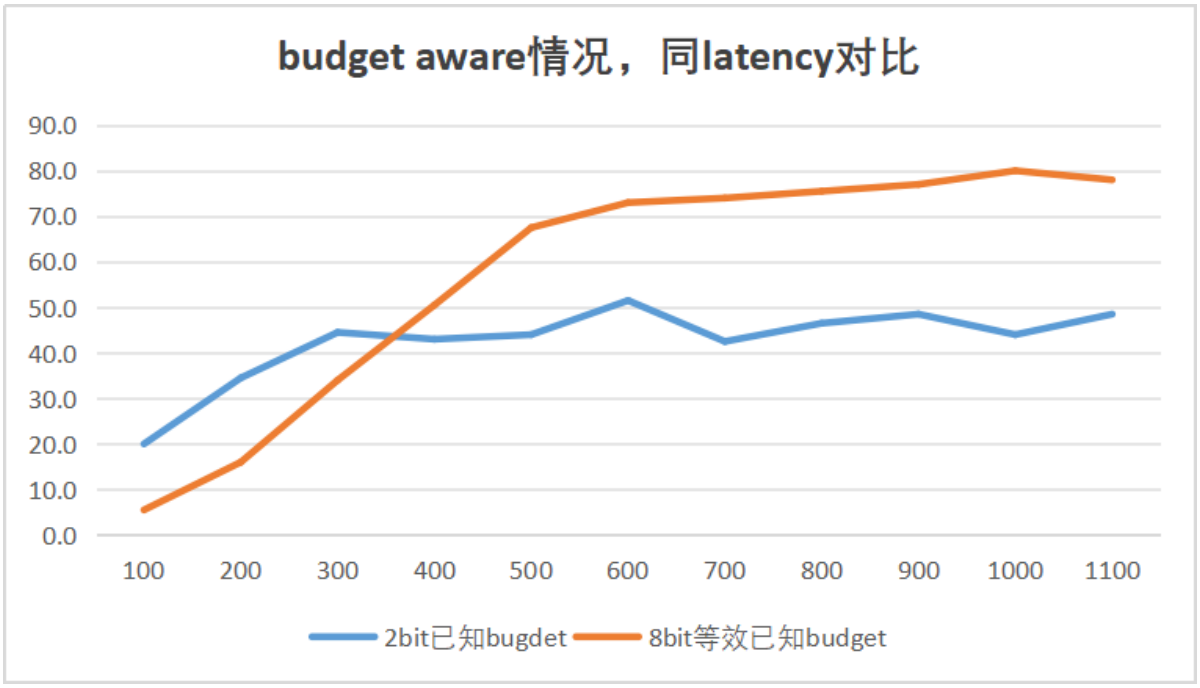
考虑到我们之前已经获取了两个模型之间的latency存在2.73的倍数关系。如在开发板的端侧环境下，2bit模型输出100个token，8bit模型在相同的时间里可以输出100/2.73≈37个token。我们下面探索这两个答案的质量。

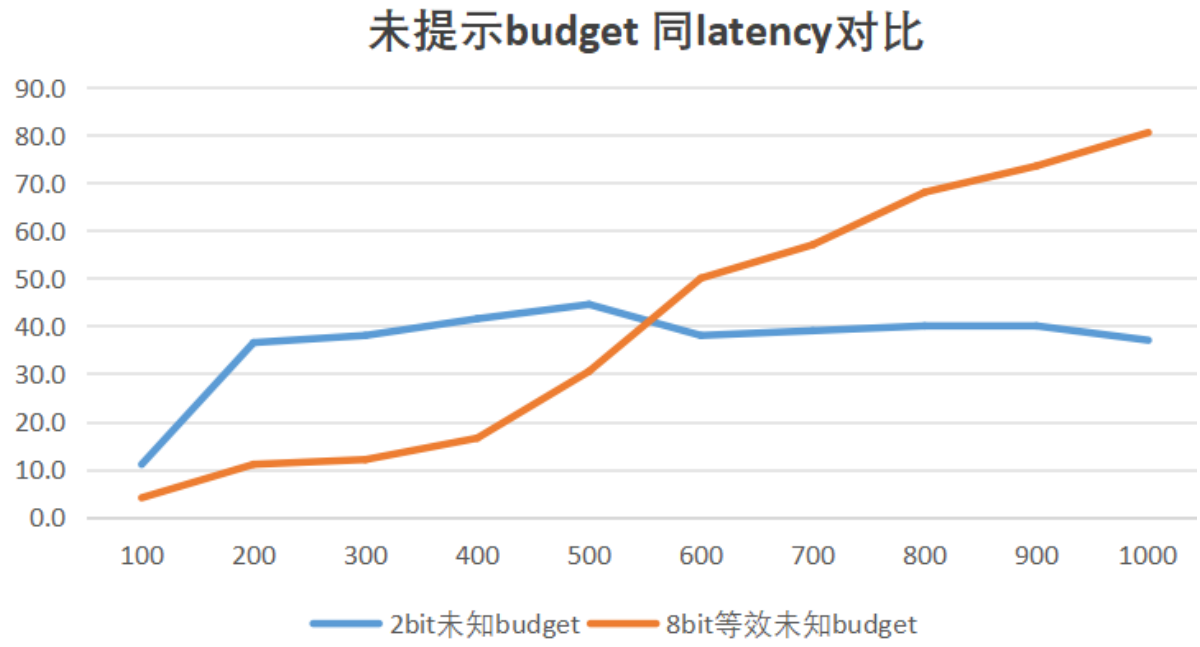
我们换算了8bit模型的横坐标，做法是将2bit数据点横坐标100 200 300……分别除以2.73。模拟同latency的情况。



然后对比了同latency下8bit和2bit的效果


正确率随时间的变化
90.0
80.0
70.0
60.0
50.0
40.0
30.0
20.0
10.0
0.0
100
200
300
400
500
600
700
800
900
1000
1100
1200
2bit已知bugdet
2bit未知budget
8bit等效已知budget
8bit等效未知budget


budget aware情况，同latency对比
90.0
80.0
70.0
60.0
50.0
40.0
30.0
20.0
10.0
0.0
100
200
300
400
500
600
700
800
900
1000
1100
2bit已知bugdet
8bit等效已知budget


未提示budget 同latency对比
90.0
80.0
70.0
60.0
50.0
40.0
30.0
20.0
10.0
0.0
100
200
300
400
500
600
700
800
900
1000
2bit未知budget
8bit等效未知budget

在latency较小时，小模型体现出优势，因为答案相对较长。在latency逐步增加时，大模型逐渐体现出优势。

在budget aware情况下，大模型可以更快的超越小模型。这可能是因为大模型可以更好的理解限制，也有更强的长度控制能力。