Atlas 200I DK A2 开发者套件
23.0.RC3
智能 ChatBot 应用开发指南

文档版本 01

发布日期 2023-11-14

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

安全声明

漏洞声明

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该政策可参考华为公司官方网站的网址：https://www.huawei.com/cn/psirt/vul-response-process。
如企业客户须获取漏洞信息，请访问：https://securitybulletin.huawei.com/enterprise/cn/security-advisory。

目 录

1 样例介绍

智能ChatBot应用是通过对ChatYuan-Large系列大型语言模型进行压缩，将模型转换为ONNX或OM模型，使其能够全部或部分运行在开发者套件的昇腾AI处理器上并加速。使用流式输出对话的方式在网页前端收集数据，进行文本解析后，通过后端放入消息队列，后端推理进程获取队列中的文本后，使用大型语言模型进行推理并返回推理结果至前端页面。最终实现与聊天机器人进行对话的场景。

图 1-1 实现原理图

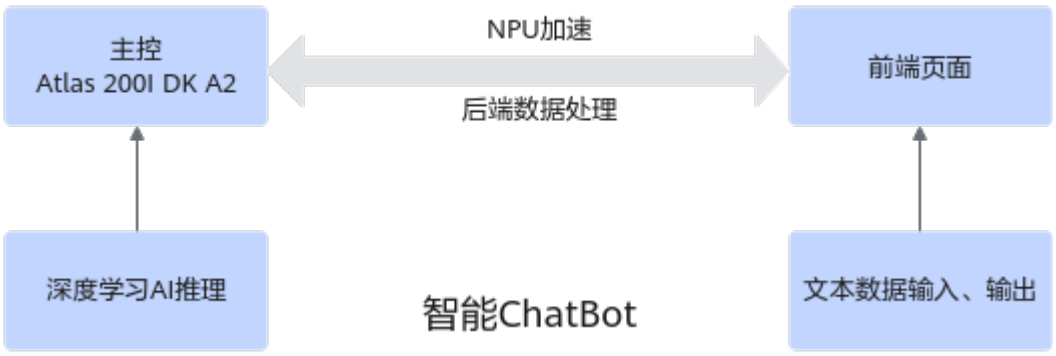
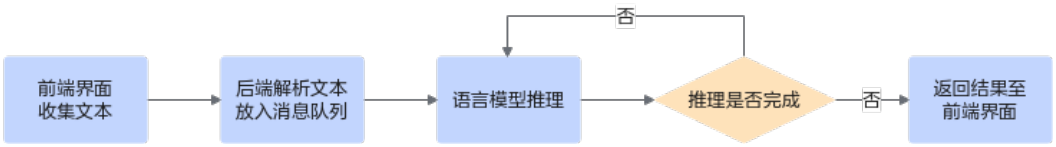


图 1-2 语音控制功能原理图



2 快速体验

环境准备

1. 参见**一键制卡**章节烧录镜像，将SD卡插入开发者套件的卡槽，**当前样例仅在Ubuntu OS适配验证过，未在openEuler OS适配验证，推荐烧录Ubuntu OS镜像。**
2. 确保开发者套件通过以太网口连接PC或者路由器（如果是连接路由器，需要确保PC也要接入路由器，且PC和开发者套件网络相通）。
3. 参见**通过PC共享网络联网（Windows）**或者**通过路由器联网**实现开发者套件连接外部网络，用于安装样例所需的依赖。

获取代码

步骤1 远程登录开发者套件，进入“/usr/local”目录运行脚本拉取代码。

```
cd /usr/local
```

步骤2 运行脚本拉取代码。

```
bash E2E_samples_download_tool.sh -d download_destination_path -s source_repository -b branch target_path
```

参数说明：

- -d：指定代码的下载路径。
- -s：指定开源仓库的clone url。
- -b：指定开源仓库分支名称及待下载的项目目录。
- -f：强制更新下载路径中的目录。当样例目录已删除，但重新下载时提示“Already up to date”时可使用此参数。

命令示例：

```
bash E2E_samples_download_tool.sh -d /home/HwHiAiUser/E2ESamples -s https://gitee.com/HUAWEI-ASCEND/ascend-devkit.git -b master src/E2E-Sample/ChatBot/
```

回显如下：

```
Download E2E samples successfully!
```

执行完成后，会在“/home/HwHiAiUser/E2Esamples”目录下生成“src/E2E-Sample/ChatBot/”目录。

步骤3 准备文件。

1. 单击**链接**下载chatbot前端相关文件，将解压后目录中文件上传至服务器“/home/HwHiAiUser/E2ESamples/src/E2E-Sample/ChatBot/Demo_V1/dist_chatbot_standalone”目录下。
2. 单击**链接**下载模型文件并解压，将解压后目录中文件上传至服务器“/home/HwHiAiUser/E2ESamples/src/E2E-Sample/ChatBot/Demo_V1/models”目录下。
3. 单击**链接**下载chabot配置文件并解压，将解压后目录中的文件上传至服务器“/home/HwHiAiUser/E2ESamples/src/E2E-Sample/ChatBot/Demo_V1/tokenizer_file”目录下

步骤4 进入智能ChatBot代码目录。

```
cd /home/HwHiAiUser/E2ESamples/src/E2E-Sample/ChatBot/Demo_V1
```

智能ChatBot应用工程目录文件详细介绍请参见**3 代码实现**。

步骤5 安装依赖。

```
pip3 install -r requirements.txt
```

----结束

关闭 OOM Killer 与图形桌面

使用swap分区时如果开启了OOM Killer，内存将会在占用3.01G时进行swap分区交换，此时ChatBot的推理预热将会失效。请参见以下步骤关闭OOM Killer

步骤1 执行以下命令关闭OOM Killer服务。

```
systemctl disable oom_killer
```

说明

若需要重新开启，请参见**配置进程内存限制（OOM Killer）**解决。

步骤2 分别将/root/.bashrc与/home/HwHiAiUser/.bashrc文件中代码注释。

- 打开/root/.bashrc文件。
  ```
  vi /root/.bashrc
  ```
- 打开/home/HwHiAiUser/.bashrc
  ```
  vi /home/HwHiAiUser/.bashrc
  ```

注释以下代码：

```
echo $$ > /sys/fs/cgroup/memory/usermemory.tasks
```

步骤3 执行以下命令关闭图形桌面。

```
systemctl disable toggle_graphical_desktop
```

步骤4 执行命令重启开发者套件。

```
reboot
```

----结束

运行智能 ChatBot 应用

步骤1 进入智能ChatBot应用工程目录。

```
cd /home/HwHiAiUser/E2ESamples/src/E2E-Sample/ChatBot/Demo_V1
```

步骤2 清除缓存并运行推理进程脚本。

```
free -h && sudo sysctl -w vm.drop_caches=3 && sudo sync && echo 3 | sudo tee /proc/sys/vm/drop_caches && free -h
```


在当前窗口执行命令运行推理进程脚本。

```
python3 generate.py
```

回显如下：

```
[INFO] acl init success
[INFO] open device 0 success
[INFO] load model ./models/encoder.om success
[INFO] create model description success
[INFO]The encoder has been initialized. Initializing the first decoder in progress.
2023-08-22 09:55:55.884550298 [E:onnxruntime:Default, env.cc:251 ThreadMain] pthread_setaffinity_np failed for thread: 55417, index: 2, mask: {3, }, error code: 22 error msg: Invalid argument. Specify the number of threads explicitly so the affinity is not set.
[INFO]The first decoder has been initialized. Initializing the second decoder in progress.
2023-08-22 09:56:03.641827137 [E:onnxruntime:Default, env.cc:251 ThreadMain] pthread_setaffinity_np failed for thread: 55422, index: 2, mask: {3, }, error code: 22 error msg: Invalid argument. Specify the number of threads explicitly so the affinity is not set.
[INFO]init finished
[INFO]ChatBot Ready
```

步骤3 重新打开远程链接窗口，进入对应目录运行智能ChatBot应用。

```
cd /home/HwHiAiUser/E2ESamples/src/E2E-Sample/ChatBot/Demo_V1
python3 main.py
```

回显如下：

```
 * Serving Flask app 'main'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.137.100:5000
```

步骤4 在计算机中打开Chrome浏览器，输入**步骤3**回显中加粗部分的URL地址进入页面，即可实现与聊天机器人进行对话。

----结束

3 代码实现

3.1 代码文件介绍

智能ChatBot应用关键文件（夹）及含义如**表3-1**所示。

表 3-1 代码文件介绍

文件（夹）名称	说明
main.py	智能ChatBot应用运行入口文件。
generate.py	智能ChatBot应用模型推理代码文件。
utils.py	智能ChatBot应用推理相关工具文件。
requirements.txt	智能ChatBot应用环境依赖文件。
dist_chatbot_standalone	智能ChatBot应用前端组件。
models	智能ChatBot应用模型文件目录。
tokenizer_file	智能ChatBot应用的分词器文件，用于为ChatBot提供语言基础。

3.2 智能 ChatBot 应用运行入口代码

“main.py”为ChatBot应用运行主入口，代码解析如下：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```
import json
import os

from filelock import FileLock
from flask import Flask, request, jsonify
from flask import render_template  # 引入模板插件
from flask_cors import CORS

temp_path = os.path.join(os.getcwd(), 'temp')
input_filepath = os.path.join(temp_path, 'input.txt')
output_filepath = os.path.join(temp_path, 'output.json')
lock_filepath = os.path.join(temp_path, 'lock.txt')
# 创建文件锁
lock = FileLock(lock_filepath, timeout=5)


# 将输入文本写入文件
def write_input(text):
    with lock:
        with open(input_filepath, 'w', encoding='utf-8') as f:
            f.write(text)


if __name__ == '__main__':
    # 创建Flask实例
    app = Flask(
        __name__,
        static_folder='./dist_chatbot_standalone',  # 设置静态文件夹目录
        template_folder="./dist_chatbot_standalone",
        static_url_path=""
    )

    # 允许跨域访问
    CORS(app, resources=r'/*')

    @app.route('/')
    def index():
        return render_template('index.html', name='index')

    @app.route("/chat", methods=["GET"])
    def getChat():
        args = request.args
        message = args.get("message")
        print('msg received')
        write_input(message)
        data = {
            "code": 200,
            "message": ''
        }
        return jsonify(data)


    @app.route("/getMsg", methods=["GET"])
    def getMsg():
        if not lock.is_locked:
            try:
                with open(output_filepath, 'r', encoding='utf-8') as f:
                    data = json.load(f)
                os.remove(output_filepath)
                print(data)
            except FileNotFoundError:
                data = {
                    "code": 200,
                    "data": {
                        "isEnd": False,
                        "message": ""
                    }
                }
        else:
```

```
            data = {
                "code": 200,
                "data": {
                    "isEnd": False,
                    "message": ""
                }
            }

        return jsonify(data)


    app.run(
        use_reloader=False,
        host="0.0.0.0",
        port=5000
    )
```

3.3 智能 ChatBot 应用模型推理代码

“generate.py”是智能ChatBot应用模型推理代码，代码解析如下：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import json
import os
import time

from filelock import FileLock
from ais_bench.infer.interface import InferSession

from utils import preprocess, postprocess, content_generate, generate_onnx_input


# 创建临时文件夹
temp_path = os.path.join(os.getcwd(), 'temp')
os.makedirs(temp_path, exist_ok=True)

input_filepath = os.path.join(temp_path, 'input.txt')
output_filepath = os.path.join(temp_path, 'output.json')
lock_filepath = os.path.join(temp_path, 'lock.txt')

# 创建文件锁
lock = FileLock(lock_filepath, timeout=5)


# 检查是否有待处理的输入文件
def check_input():
    with lock:
        try:
            with open(input_filepath, 'r', encoding='utf-8') as f:
                text = f.read()
            # 删除或清空文件内容
            os.remove(input_filepath)
            return text
        except FileNotFoundError:
            return None


# 将输出结果写入文件
def write_output_json(data):
    with lock:
        while os.path.exists(output_filepath):
            time.sleep(0.1)
        with open(output_filepath, 'w', encoding='utf-8') as f:
            json.dump(data, f, ensure_ascii=False, indent=4)
```



```
# 预先加载om模型确保GE能申请到足够的内存
encoder = InferSession(0, './models/encoder.om')

if __name__ == '__main__':
    # 确保模型加载完成后再导入其他包
    import numpy as np
    import onnxruntime
    import torch
    from transformers import T5Tokenizer
    from transformers.generation import LogitsProcessorList, NoRepeatNGramLogitsProcessor,
TemperatureLogitsWarper, \
        TopKLogitsWarper, StoppingCriteriaList, MaxLengthCriteria

    print('[INFO]The encoder has been initialized. Initializing the first decoder in progress.')
    first_decoder = onnxruntime.InferenceSession('./models/decoder_first_sim_quant.onnx')
    print('[INFO]The first decoder has been initialized. Initializing the second decoder in progress.')
    decoder_iter = onnxruntime.InferenceSession('./models/decoder_iter_sim_quant.onnx')
    tokenizer = T5Tokenizer.from_pretrained("./tokenizer_file")
    dummy_decoder_input_ids = np.array([[0]], dtype=np.int64)
    logits_processor = LogitsProcessorList([NoRepeatNGramLogitsProcessor(3)])
    logits_warper = LogitsProcessorList(
        [TemperatureLogitsWarper(0.7), TopKLogitsWarper(filter_value=float('-inf'), top_k=50)])
    stopping_criteria = StoppingCriteriaList([MaxLengthCriteria(512)])
    eos_token_id = [1]
    record = []
    print('[INFO]init finished')


    def generate_ss_mode(input_text, output=False):
        if input_text == 'clear' and output:
            print('record clear')
            record.clear()
            data = {
                "code": 200,
                "data": {
                    "isEnd": False,
                    "message": '聊天记录已清空'
                }
            }
            write_output_json(data)
            return

        # 生成附带上下文的模型输入
        content = content_generate(record, input_text)

        # 对输入文本进行预处理，生成token和attention_mask
        inputs = tokenizer(text=[preprocess(content)], truncation=True, padding='max_length',
max_length=768,
                           return_tensors="np")

        encoder_input_ids = inputs['input_ids']
        attention_mask = inputs['attention_mask']

        # 使用encoder模型生成encoder_hidden_states
        encoder_hidden_states = encoder.infer([encoder_input_ids, attention_mask])[0]

        print('autoregression start')
        first_loop = True
        decoder_input_ids = dummy_decoder_input_ids

        input_ids = torch.tensor(dummy_decoder_input_ids)
        unfinished_sequences = torch.tensor([1])
        while True:
            if first_loop:
                outputs = first_decoder.run(None, {'decoder_input_ids': decoder_input_ids,
                                                   'hidden_states': encoder_hidden_states,
                                                   'attention_mask': attention_mask})
                first_loop = False
            else:
```

```
            onnx_input = generate_onnx_input(decoder_input_ids, attention_mask, past_key_values)
            outputs = decoder_iter.run(None, onnx_input)
        logits = torch.tensor(outputs[0])
        past_key_values = outputs[1:]
        next_token_logits = logits[:, -1, :]

        next_token_scores = logits_processor(input_ids, next_token_logits)
        next_token_scores = logits_warper(input_ids, next_token_scores)

        probs = torch.nn.functional.softmax(next_token_scores, dim=-1)
        next_tokens = torch.multinomial(probs, num_samples=1).squeeze(1)
        message = postprocess(tokenizer.batch_decode(next_tokens, skip_special_tokens=True)[0])
        if message == ' ' or message == '':
            message = ' '
        elif message == '\n':
            message = '<br />'

        data = {
            "code": 200,
            "data": {
                "isEnd": False,
                "message": message
            }
        }

        input_ids = torch.cat([input_ids, next_tokens[:, None]], dim=-1)
        decoder_input_ids = input_ids[:, -1:].numpy()

        # 判断是否结束
        unfinished_sequences = unfinished_sequences.mul((sum(next_tokens != i for i in
eos_token_id)).long())
        if unfinished_sequences.max() == 0 or stopping_criteria(input_ids, None):
            data['data']['isEnd'] = True
            if output:
                write_output_json(data)

            break
        else:
            if output:
                write_output_json(data)

    out_text = tokenizer.batch_decode(input_ids, skip_special_tokens=True)[0]
    out_text = postprocess(out_text)

    record.append([input_text, out_text])
    if tokenizer(text=[preprocess(content_generate(record, ''))], truncation=True, padding=True,
                 max_length=768,
                 return_tensors="np")['attention_mask'].shape[1] > 256:
        print('record clear')
        record.clear()


print('[INFO]ChatBot Ready')
while True:
    msg = check_input()
    if msg is None:
        continue
    print('input text: ', msg)
    generate_ss_mode(msg, True)
```

3.4 智能 ChatBot 应用推理相关工具代码

“utils.py”是智能ChatBot应用推理相关工具代码主要对字符串做前后处理，对模型的输入做处理，代码解析如下：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```



```
def content_generate(records, inputs):
    if not records:
        context = ''
    else:
        context = "\n".join([f"用户：{input_text}\n小元：{ans_text}" for input_text, ans_text in records]) + "\n"
    return context + "用户：" + inputs + "\n小元："


def preprocess(t):
    return t.replace("\n", "\\n").replace("\t", "\\t")


def postprocess(text):
    return text.replace("\\n", "\n").replace("\\t", "\t")


def generate_onnx_input(decode_input_ids, attention_mask, past_key_values):
    input_list = [decode_input_ids, None, attention_mask]
    for tensor in past_key_values:
        input_list.append(tensor)

    input_dict = {'decoder_input_ids': input_list[0], 'encoder_attention_mask': input_list[2],
                  'kv_in_0_0': input_list[3], 'kv_in_0_1': input_list[4], 'kv_in_0_2': input_list[5],
                  'kv_in_0_3': input_list[6],
                  'kv_in_1_0': input_list[7], 'kv_in_1_1': input_list[8], 'kv_in_1_2': input_list[9],
                  'kv_in_1_3': input_list[10],
                  'kv_in_2_0': input_list[11], 'kv_in_2_1': input_list[12], 'kv_in_2_2': input_list[13],
                  'kv_in_2_3': input_list[14],
                  'kv_in_3_0': input_list[15], 'kv_in_3_1': input_list[16], 'kv_in_3_2': input_list[17],
                  'kv_in_3_3': input_list[18],
                  'kv_in_4_0': input_list[19], 'kv_in_4_1': input_list[20], 'kv_in_4_2': input_list[21],
                  'kv_in_4_3': input_list[22],
                  'kv_in_5_0': input_list[23], 'kv_in_5_1': input_list[24], 'kv_in_5_2': input_list[25],
                  'kv_in_5_3': input_list[26],
                  'kv_in_6_0': input_list[27], 'kv_in_6_1': input_list[28], 'kv_in_6_2': input_list[29],
                  'kv_in_6_3': input_list[30],
                  'kv_in_7_0': input_list[31], 'kv_in_7_1': input_list[32], 'kv_in_7_2': input_list[33],
                  'kv_in_7_3': input_list[34],
                  'kv_in_8_0': input_list[35], 'kv_in_8_1': input_list[36], 'kv_in_8_2': input_list[37],
                  'kv_in_8_3': input_list[38],
                  'kv_in_9_0': input_list[39], 'kv_in_9_1': input_list[40], 'kv_in_9_2': input_list[41],
                  'kv_in_9_3': input_list[42],
                  'kv_in_10_0': input_list[43], 'kv_in_10_1': input_list[44], 'kv_in_10_2': input_list[45],
                  'kv_in_10_3': input_list[46],
                  'kv_in_11_0': input_list[47], 'kv_in_11_1': input_list[48], 'kv_in_11_2': input_list[49],
                  'kv_in_11_3': input_list[50],
                  'kv_in_12_0': input_list[51], 'kv_in_12_1': input_list[52], 'kv_in_12_2': input_list[53],
                  'kv_in_12_3': input_list[54],
                  'kv_in_13_0': input_list[55], 'kv_in_13_1': input_list[56], 'kv_in_13_2': input_list[57],
                  'kv_in_13_3': input_list[58],
                  'kv_in_14_0': input_list[59], 'kv_in_14_1': input_list[60], 'kv_in_14_2': input_list[61],
                  'kv_in_14_3': input_list[62],
                  'kv_in_15_0': input_list[63], 'kv_in_15_1': input_list[64], 'kv_in_15_2': input_list[65],
                  'kv_in_15_3': input_list[66],
                  'kv_in_16_0': input_list[67], 'kv_in_16_1': input_list[68], 'kv_in_16_2': input_list[69],
                  'kv_in_16_3': input_list[70],
                  'kv_in_17_0': input_list[71], 'kv_in_17_1': input_list[72], 'kv_in_17_2': input_list[73],
                  'kv_in_17_3': input_list[74],
                  'kv_in_18_0': input_list[75], 'kv_in_18_1': input_list[76], 'kv_in_18_2': input_list[77],
                  'kv_in_18_3': input_list[78],
                  'kv_in_19_0': input_list[79], 'kv_in_19_1': input_list[80], 'kv_in_19_2': input_list[81],
                  'kv_in_19_3': input_list[82],
                  'kv_in_20_0': input_list[83], 'kv_in_20_1': input_list[84], 'kv_in_20_2': input_list[85],
                  'kv_in_20_3': input_list[86],
                  'kv_in_21_0': input_list[87], 'kv_in_21_1': input_list[88], 'kv_in_21_2': input_list[89],
                  'kv_in_21_3': input_list[90],
                  'kv_in_22_0': input_list[91], 'kv_in_22_1': input_list[92], 'kv_in_22_2': input_list[93],
                  'kv_in_22_3': input_list[94],
```



```
                'kv_in_23_0': input_list[95], 'kv_in_23_1': input_list[96], 'kv_in_23_2': input_list[97],
                'kv_in_23_3': input_list[98]}
    return input_dict
```

