FPGA 编译项目报告

——技术报告

高琦，苏灿，黎睿正

1. 项目背景

目前大模型部署存在的难点

- 工具结合度低

  现有的量化、打分和部署等工具模块间甚至模块内操作逻辑不一致，许多操作之间仍需手动调整参数。

- 新模型适配成本高

  现有的工作流需要根据所使用的模型手动调整，新模型需要逐一环节调试，无法开箱即用。

- 操作学习成本高

  现有的工具多为命令行工具，缺少直观的交互逻辑和图形化界面，用户操作上手难度较大。

2. 项目内容

2.1 技术特性

- 量化：支持 GPTQ 指定 bits、group_size、desc_act
- 打分
  - 支持 lm-evaluation-harness 测试项

    arc_easy、arc_challenge、gsm8k_cot、gsm8k_platinum_cot、hellaswag、mmlu、gpqa、boolq、openbookqa
  - 支持 EvalPlus 测试项

    humaneval、mbpp
- GPU 部署：支持 vLLM 指定上下文长度、显存占用限制、服务端口、API 密钥
- 权重处理：支持 Compiler-VCU128 指定 bits、group_size、desc_act

- FPGA 服务：支持 Fast API 指定上下文长度、生成温度、服务端口、API 密钥
- GPU、Port 调度器：支持 GPU 设备图分类、调整调度显存占用量、设备锁机制
- WebUI：支持指定服务地址、用户管理

2.2 WebUI

前端 WebUI 主要支持用户进行一键式操作，为大模型上板做好准备工作，提供用户登录、各项部署细节选择、取消进程、实时日志查看服务。

- 网页界面展示

图 1 用户登录界面展示

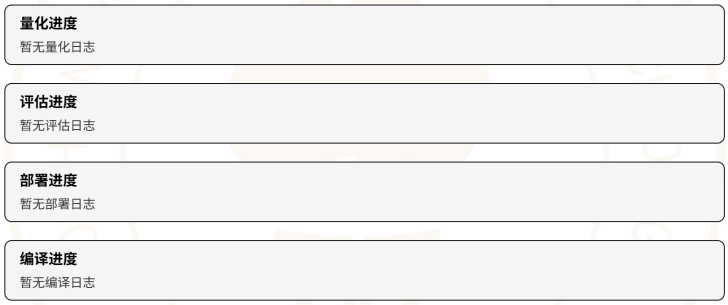

图 2 日志界面展示

图 3 部署工具界面展示

```
    //工具选项设置代码
data() {
    return {
      activeSections: ['model', 'quant', 'target', 'eval', 'task'],
      selectedModel: '',
      selectedQuantPrecision: 'int4',
      isDeploying: false,
      deployStatus: [],
      pollingInterval: null,
      progressPollingInterval: null,
      quantPid: null,
      selectedEvalMethod: 'evalPlus',
      selectedEvalTarget: 'none',
      quantLogs: [],
      evalLogs:[],
      dataLogs:[],
      models: [
        { value: 'qwen2', label: 'Qwen2-7B-Instruct', icon: modelQwen
            },
        { value: 'qwen2.5', label: 'Qwen2.5-7B-Instruct', icon:
            modelQwen },
        { value: 'qwen2-vl', label: 'Qwen2-VL-7B-Instruct', icon:
```

```
            modelQwen },
          { value: 'qwen2.5-vl', label: 'Qwen2.5-VL-7B-Instruct', icon:
            modelQwen },
          { value: 'deepseek', label: 'DeepSeek-R1-Distill-Qwen-7B', icon
            : modelDeepseek }
        ],
        precisions: [
          { value: 'int2', label: 'INT2', precisionValue: 2 },
          { value: 'int4', label: 'INT4 (仅支持) ', precisionValue: 4 },
          { value: 'int8', label: 'INT8', precisionValue: 8 }
        ],
        evalMethods: [
          { label: 'EvalPlus', value: 'evalPlus' },
          { label: 'lmEvaluationHarness', value: 'lmEvalHarness' }
        ],
        evalPlusTasks: [
          { value: 'humaneval', label: 'HumanEval' },
          { value: 'mbpp', label: 'MBPP' }
        ],
        lmEvalHarnessTasks: [
          { value: 'arc_easy', label: 'ARC Easy' },
          { value: 'arc_challenge', label: 'ARC Challenge' },
          { value: 'gsm8k_cot', label: 'GSM8K CoT' },
          { value: 'gsm8k_platinum_cot', label: 'GSM8K Platinum CoT' },
          { value: 'hellaswag', label: 'HellaSwag' },
          { value: 'mmlu', label: 'MMLU' },
          { value: 'gpqa', label: 'GPQA' },
          { value: 'boolq', label: 'BoolQ' },
          { value: 'openbookqa', label: 'OpenBookQA' }
        ],
        selectedEvalTasks: [],
        evalTargets: [
          { label: '原模型', value: 'origin' },
          { label: '量化模型', value: 'quant' },
          { label: '两个都评分', value: 'both' },
          { label: '不评分', value: 'none' }
        ],
      }
  },
```

- 用户管理：支持自定义登录用户

```
  //前端登录界面代码
```

```
<script>
import axios from 'axios'

export default {
  name: 'Login',
  data() {
    return {
      loading: false,
      loginForm: {
        username: '',
        password: ''
      },
      rules: {
        username: [
          { required: true, message: '请输入用户名', trigger: 'blur' }
        ],
        password: [
          { required: true, message: '请输入密码', trigger: 'blur' }
        ]
      }
    }
  },
  methods: {
    handleLogin() {
      this.$refs.loginForm.validate(valid => {
        if (valid) {
          this.loading = true
          this.verifyCredentials()
        }
      })
    },
    async verifyCredentials() {
      try {
        const authHeader = 'Basic ' + btoa(`${this.loginForm.username}:
            ${this.loginForm.password}`)
        const response = await axios.get('/api/verify', {
          headers: { 'Authorization': authHeader }
        })

        if (response.data.success) {
          this.$emit('login-success', {
            username: this.loginForm.username,
            password: this.loginForm.password
```

```
          })
        } else {
          throw new Error(response.data.message || '认证失败')
        }
      } catch (error) {
        let errorMsg = '登录失败: '
        if (error.response) {
          errorMsg += error.response.data?.message || error.response.
              statusText
        } else {
          errorMsg += error.message
        }
        this.$message.error(errorMsg)
        console.error('登录错误详情:', error)
      } finally {
        this.loading = false
      }
    }
  }
}
</script>
```

提供用户登录界面，使用体验更规范，同时可根据不同账户指定特定权限如管理员权限。支持客户添加指定账户，后续可拓展使用实时 token 验证方式提高安全性。

- 打分：支持同时开展多框架多测试项测试和原模型、量化模型对比测试

```
//轮询日志同步进度信息
async startEvaluationPolling(target) {
  let failCount = 0;
  const MAX_FAILS = 5;

  const interval = setInterval(async () => {
    try {
      const response = await axios.get(`${this.apiUrl}/eval_progress`,
          {
        headers: {
          'Authorization': 'Basic ' + btoa(`${this.authInfo.username}:${
              this.authInfo.password}`)
        }
      });

      if (response.data.success) {
```

```
        failCount = 0;
        const progressLines = response.data.progress || [];
        this.evalLogs.push(...progressLines);
        this.$emit('eval-log', progressLines);

        const hasError = this.evalLogs.some(line =>
            line.includes('[ERROR]') ||
            line.includes('失败') ||
            line.includes('异常') ||
            line.includes('Traceback')
        );

        const hasCompleted = this.evalLogs.some(line =>
            line.includes('完成') ||
            line.toLowerCase().includes('evaluation finished') ||
            line.toLowerCase().includes('scoring complete') ||
            line.toLowerCase().includes('done')
        );

        if (hasError) {
          clearInterval(interval);
          this.deployStatus.push(` ${target === 'origin' ? '原模型' : '
              量化模型'} 评分失败，请检查日志`);
          return;
        }

        if (!response.data.is_running && !hasCompleted) {
          clearInterval(interval);
          this.deployStatus.push(` ${target === 'origin' ? '原模型' : '
              量化模型'} 评分中断但未检测到“完成”关键词，可能失败`);
          return;
        }

        if (hasCompleted) {
          clearInterval(interval);
          this.deployStatus.push(` ${target === 'origin' ? '原模型' : '
              量化模型'} 评分完成`);
        }
      }
    } catch (error) {
      const isAxiosError = error.isAxiosError;
      const errMsg = error?.message || '';
      const isIgnorable = errMsg.includes('ERR_EMPTY_RESPONSE') ||
```

```
          (isAxiosError && !error.response);

        if (isIgnorable) {
          failCount++;
          console.warn(` 评分轮询失败（可忽略）: ${errMsg}, 当前失败次数: $
              {failCount}`);

          if (failCount >= MAX_FAILS) {
            clearInterval(interval);
            this.deployStatus.push(` ${target === 'origin' ? '原模型' : '
                量化模型'} 连续多次无法获取评分进度，任务可能失败`);
            this.$message.error('评分进度查询连续失败，已中止');
          }
          return;
        }
        clearInterval(interval);
        this.deployStatus.push(` ${target === 'origin' ? '原模型' : '量化
            模型'} 评分进度获取失败: ${errMsg}`);
        this.$message.error('评分进度查询失败');
      }
    }, 3000);
  },
```

当流程进行到原模型打分或量化模型评分时调用 startEvaluation 启动评分程序

```
  //评分部分开始
  async startEvaluation(target) {
    const model = this.getCurrentModel();
    const method = this.selectedEvalMethod;

    this.evalLogs = [];
    this.deployStatus.push(`开始对 ${target === 'origin' ? '原模型' : '量
        化模型'} 进行评分（方法: ${method}）...`);

    try {
      const response = await axios.post(`${this.apiUrl}`, {
        model_name: model.label,
        eval_method: method,
        eval_tasks: this.selectedEvalTasks,
        start_evaluation: true,
        is_quantized: target !== 'origin'
      }, {
        headers: {
          'Authorization': 'Basic ' + btoa(`${this.authInfo.username}:${
```

```
            this.authInfo.password}`)
        }
      });

      if (response.data.success) {
        this.deployStatus.push(` ${target === 'origin' ? '原模型' : '量化
            模型'} 评分任务已启动`);
        this.startEvaluationPolling(target);
      } else {
        throw new Error(response.data.message || '评分启动失败');
      }
    } catch (error) {
      console.error(`评分启动失败 (${target})`, error);
      const errorMsg = error.response?.data?.message || error.message;
      this.deployStatus.push(` ${target === 'origin' ? '原模型' : '量化模
          型'} 评分启动失败: ${error.message}`);
    }
  },
```

当用户取消部署时，前端向 api 发送取消信息，后端分别取消对应流程代码

```
  //前端向api发送取消信息并监控是否成功取消进程
  async cancelDeploy() {
    try {
      if (!this.isDeploying) return;

      if (this.progressPollingInterval) {
        clearInterval(this.progressPollingInterval);
      }

      this.deployStatus.push('正在取消部署流程...');

      try {
        const quantCancelResp = await axios.post(`${this.apiUrl}/
            cancel_quant`, {}, {
          headers: {
            'Authorization': 'Basic ' + btoa(`${this.authInfo.username}:${
                this.authInfo.password}`)
          }
        });

        if (quantCancelResp.data.success) {
          this.deployStatus.push('  已成功取消量化进程');
        } else {
```

```
    this.deployStatus.push(' 取消量化失败: ' + quantCancelResp.data.
        message);
  }
} catch (e) {
  this.deployStatus.push(' 取消量化时发生异常: ' + (e.message));
}

try {
  const cancelResp = await axios.post(`${this.apiUrl}/cancel_eval`,
      {}, {
    headers: {
      'Authorization': 'Basic ' + btoa(`${this.authInfo.username}:${
          this.authInfo.password}`)
    }
  });

  if (cancelResp.data.success) {
    this.deployStatus.push(` 已取消评分进程`);
  } else {
    this.deployStatus.push(` 无法取消评分: ${cancelResp.data.message
        }`);
  }
} catch (error) {
  this.deployStatus.push(` 取消评分失败: ${error.message}`);
}

try {
  const cancelResp = await axios.post(`${this.apiUrl}/
      cancel_deployment`, {}, {
    headers: {
      'Authorization': 'Basic ' + btoa(`${this.authInfo.username}:${
          this.authInfo.password}`)
    }
  });

  if (cancelResp.data.success) {
    this.deployStatus.push(` 已取消部署进程`);
    this.$message.success('部署进程已成功取消');
  } else {
    this.deployStatus.push(` 无法取消部署: ${cancelResp.data.message
        }`);
    this.$message.warning(cancelResp.data.message || '无法取消部署进
        程');
```

```
      }
    } catch (error) {
      const errMsg = error?.response?.data?.message || error.message;
      this.deployStatus.push(` 取消部署失败: ${errMsg}`);
      this.$message.error('取消部署失败');
    }

    try {
      const cancelResp = await axios.post(`${this.apiUrl}/
          cancel_compile`, {}, {
        headers: {
          'Authorization': 'Basic ' + btoa(`${this.authInfo.username}:${
              this.authInfo.password}`)
        }
      });

      if (cancelResp.data.success) {
        this.deployStatus.push(` 已取消编译进程`);
      } else {
        this.deployStatus.push(` 无法取消编译: ${cancelResp.data.message
            }`);
      }
    } catch (error) {
      this.deployStatus.push(` 取消编译失败: ${error.message}`);
    }

    this.isDeploying = false;
    this.$message.warning('部署流程和评分流程已中断');

  } catch (error) {
    console.error('取消部署失败:', error);
    const errorMsg = error.response?.data?.message || error.message;
    this.deployStatus.push(` 取消失败: ${errorMsg}`);
    this.$message.error(`取消失败: ${errorMsg}`);
  } finally {
    this.isDeploying = false;
  }
},
```

```
//取消评分进程
@app.route('/api/cancel_eval', methods=['POST'])
@auth.login_required
def cancel_evaluation():
```

```
    global current_eval_process

    try:
        if not is_eval_running():
            return jsonify({
                'success': False,
                'message': '没有正在运行的评估进程'
            }), 400

        with open(EVALUATION_LOG, 'a') as f:
            f.write("[INFO] 正在取消评估进程...\n")

        current_eval_process.terminate()
        current_eval_process.join(timeout=2)

        with open(EVALUATION_LOG, 'a') as f:
            f.write("[INFO] 评估进程已被用户取消\n")

        current_eval_process = None

        return jsonify({
            'success': True,
            'message': '评估进程已成功取消'
        })
    except Exception as e:
        error_msg = f"取消评估失败: {str(e)}"
        log_error(error_msg, "backend")
        return jsonify({
            'success': False,
            'message': error_msg
        }), 500
```

- GPU 部署：支持同时部署原模型、量化模型以便对比

```
//根据用户需求进行对应模型部署
async startDeployment() {
  const model = this.getCurrentModel();

  this.deployLogs = [];
  this.deployStatus.push(`开始部署模型 ${model.label} ...`);

  try {
    const response = await axios.post(`${this.apiUrl}`, {
```

```
        model_name: model.label,
        start_deployment: true
      }, {
        headers: {
          'Authorization': 'Basic ' + btoa(`${this.authInfo.username}:${
              this.authInfo.password}`)
        }
      });

      if (response.data.success) {
        this.deployStatus.push(` 模型 ${model.label} 部署任务已启动`);
        this.startDeploymentPolling();
      } else {
        throw new Error(response.data.message || '部署启动失败');
      }
    } catch (error) {
      console.error('部署启动失败', error);
      const errorMsg = error.response?.data?.message || error.message;
      this.deployStatus.push(` 模型 ${model.label} 部署启动失败: ${
          errorMsg}`);
    }
  },
```

用户可在模型部署界面选择对原模型/量化模型进行部署，后续将补充对话框，实现部署后用户即时与模型互动直观体验功能

```
//模型部署界面
<template>
  <el-card class="deployment-tool-card">
    <div slot="header" class="header-with-icon">
      <img src="@/assets/header-icon.jpg" class="header-icon">
      <span>部署模型管理</span>
    </div>

    <el-form label-position="top">
      <!-- 模型名称选择 -->
      <el-form-item label="模型名称">
        <el-select v-model="selectedModel" placeholder="请选择模型">
          <el-option
              v-for="model in models"
              :key="model.value"
              :label="model.label"
              :value="model.value">
```

```
            <span style="float:␣left">{{ model.label }}</span>
            <img :src="model.icon" class="option-icon" />
          </el-option>
        </el-select>
      </el-form-item>

      <!-- 部署类型选择 -->
      <el-form-item label="部署类型">
        <el-select v-model="deployType" placeholder="请选择部署类型">
          <el-option
              v-for="type␣in␣deployTypes"
              :key="type.value"
              :label="type.label"
              :value="type.value">
            <span style="float:␣left">{{ type.label }}</span>
          </el-option>
        </el-select>
      </el-form-item>

      <!-- 启动部署按钮 -->
      <div class="deploy-button-wrapper">
        <el-form-item>
          <el-button type="primary"
                     :loading="isDeploying"
                     :disabled="!selectedModel␣||␣!deployType"
                     @click="startDeploy">
            启动部署
          </el-button>
        </el-form-item>
      </div>

      <!-- 部署日志展示区域 -->
      <el-card class="deploy-log-card" v-if="deployLogs.length␣>␣0">
        <div class="log-title">部署日志</div>
        <div class="log-content">
          <div v-for="(log,␣index)␣in␣deployLogs" :key="index" class="
              log-line">{{ log }}</div>
        </div>
      </el-card>

    </el-form>
  </el-card>
</template>
```

- API 接口：捕捉用户需求并启动后端程序

```
WORKSPACE_ROOT = "/data/disk0/Workspace/Compiler-Toolchain/Compiler-
    Toolchain"
sys.path.insert(0, WORKSPACE_ROOT)
os.chdir(WORKSPACE_ROOT)
//以量化板块为例展示进程启动与日志捕获
def quantification_entrypoint(model_id, log_path, is_vl_model):
try:
    gptq_log_dir = os.path.join(WORKSPACE_ROOT, "CT", "WebUI", "
        gptq_log")
    os.makedirs(gptq_log_dir, exist_ok=True)
    os.chdir(gptq_log_dir)

    with open(log_path, 'a') as f:
        with contextlib.redirect_stdout(f), contextlib.redirect_stderr(
            f):
            if is_vl_model:
                from CT.Example.Quantization.qwenVLQuantization import
                    simpleQuantization
                print(f"[INFO] 启动VL模型量化: {model_id}")
            else:
                from CT.Example.Quantization.quantization import
                    simpleQuantization
                print(f"[INFO] 启动普通模型量化: {model_id}")

            simpleQuantization(model_id)

            print(f"[INFO] 模型量化完成: {model_id}")

except Exception as e:
    with open(log_path, 'a') as f:
        f.write(f"[ERROR] 模型量化异常: {e}\n")

def is_quant_running():
    global current_quant_process
    return current_quant_process is not None and current_quant_process.
        is_alive()

def run_quantification(model_name):
    global current_quant_process
```

```
    try:
        # 清空进度日志
        with open(PROGRESS_LOG, 'w') as f:
            f.write("")

        is_vl_model = "VL" in model_name.upper()
        log_error(f"开始量化模型:␣{model_name}␣(类型:␣{'VL'␣if␣
            is_vl_model␣else␣'普通'})", "quant")

        # 创建并启动量化进程
        current_quant_process = multiprocessing.Process(
            target=quantification_entrypoint,
            args=(model_name, PROGRESS_LOG, is_vl_model,)
        )
        current_quant_process.start()

        return current_quant_process.pid

    except Exception as e:
        error_msg = f"量化失败␣-␣模型:{model_name}␣错误:{traceback.
            format_exc()}"
        log_error(error_msg, "quant")
        current_quant_process = None
        raise
```

```
//接收前端信息执行对应操作
@app.route('/api', methods=['POST'])
@auth.login_required
def post_api():
    try:
        if data.get("start_quantization"):
        if "model_name" not in data:
            error_msg = "缺少模型名称参数"
            log_error(error_msg, "backend")
            return jsonify({'success': False, 'message': error_msg}),
                400

        try:
            if is_quant_running():
                current_quant_process.terminate()
                time.sleep(1)

            pid = run_quantification(data["model_name"])
```

```
            log_error(f"已启动量化进程␣PID:␣{pid}", "backend")
            return jsonify({
                'success': True,
                'message': '量化进程已启动',
                'pid': pid,
                'current_params': {
                    'model_name': data["model_name"]
                }
            })
        except Exception as e:
            error_msg = f"量化进程启动失败:␣{str(e)}"
            log_error(error_msg, "backend")
            return jsonify({'success': False, 'message': error_msg}),
                500

@app.route('/api/progress', methods=['GET'])
@auth.login_required
def get_progress():
    """获取量化进度"""
    try:
        if not os.path.exists(PROGRESS_LOG):
            return jsonify({
                'success': False,
                'message': '进度文件不存在',
                'is_running': False
            }), 404

        with open(PROGRESS_LOG, 'r') as f:
            lines = f.readlines()[-150:]

        # 过滤 ANSI 转义字符
        clean_lines = [remove_ansi_codes(line.strip()) for line in
            lines if line.strip()]

        return jsonify({
            'success': True,
            'progress': clean_lines,
            'is_running': is_quant_running()
        })
    except Exception as e:
        log_error(f"获取进度失败:␣{str(e)}", "backend")
        return jsonify({
```

```
            'success': False,
            'message': '获取进度失败',
            'is_running': False
        }), 500

@app.route('/api/cancel_quant', methods=['POST'])
@auth.login_required
def cancel_quantization():
    global current_quant_process

    try:
        if not is_quant_running():
            return jsonify({
                'success': False,
                'message': '没有正在运行的量化进程'
            }), 400

        with open(PROGRESS_LOG, 'a') as f:
            f.write("[INFO] 正在取消量化进程...\n")

        current_quant_process.terminate()
        current_quant_process.join(timeout=2)

        with open(PROGRESS_LOG, 'a') as f:
            f.write("[INFO] 量化进程已被用户取消\n")

        current_quant_process = None

        return jsonify({
            'success': True,
            'message': '量化进程已成功取消'
        })
    except Exception as e:
        error_msg = f"取消量化失败: {str(e)}"
        log_error(error_msg, "backend")
        return jsonify({
            'success': False,
            'message': error_msg
        }), 500
```

- 客制化：支持定制页面图标、企业名称、语言前端代码通过组件进行组合，针对客户指定的页面需求可快速调整背景图片、图标、网页风格等一系列元素，实

现便捷定制。

2.3 FPGA OpenAI-Style API

- OpenAI-Style API 提供了快捷的与 FPGA 部署模型对话的工具, 在前端 openwebui 或是 agent 直接通过 OpenAI api chat/completion 端口均可拉起工具

- openwebui 界面展示

图 4 FPGA 模型对话界面展示

- socket 实现与上位机的通信，在指定端口发送和接收消息

```
class FPGAConnection:
def __init__(self):
    self.conn = None
    self.shutdown_event = threading.Event()
    self.lock = threading.Lock()

def connect(self):

    while not self.shutdown_event.is_set():
        try:
            print(f"[H100]␣尝试连接上位机␣{TCP_HOST}:{TCP_PORT}")
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.settimeout(10)
            sock.connect((TCP_HOST, TCP_PORT))
            self.conn = sock
            print(f"[H100]␣成功连接到上位机")
            return True
        except Exception as e:
            print(f"[H100]␣连接失败:␣{str(e)}")
            time.sleep(RECONNECT_DELAY)
```

```
        return False

    def send_message(self, message):

        if self.conn is None:
            raise ConnectionError("未连接到上位机")

        try:
            with self.lock:

                full_message = message + END_MARKER.decode('utf-8')
                encoded_msg = full_message.encode('utf-8')


                msg_length = len(encoded_msg)
                self.conn.sendall(struct.pack('>I', msg_length))


                self.conn.sendall(encoded_msg)
                print(f"[H100]␣发送消息到上位机␣(长度:␣{msg_length})")

                time.sleep(2)


                length_data = self.conn.recv(4)
                if not length_data:
                    raise ConnectionError("连接已关闭")
                msg_length = struct.unpack('>I', length_data)[0]


                chunks = []
                bytes_received = 0
                while bytes_received < msg_length:
                    chunk = self.conn.recv(min(msg_length -
                        bytes_received, BUFFER_SIZE))
                    if not chunk:
                        raise ConnectionError("连接中断")
                    chunks.append(chunk)
                    bytes_received += len(chunk)

                full_message = b''.join(chunks).decode('utf-8')
```

```
                if full_message.endswith(END_MARKER.decode('utf-8')):
                    full_message = full_message[:-len(END_MARKER)]

                print(f"[H100] 收到上位机响应 (长度: {len(full_message)})
                    ")
                return full_message

            except Exception as e:
                print(f"[H100] 通信错误: {str(e)}")
                self.close()
                raise

        def close(self):
            """关闭连接"""
            if self.conn:
                try:
                    self.conn.close()
                except:
                    pass
                self.conn = None
            print("[H100] 连接已关闭")
```

- FPGA 回传消息通过转化层处理为标准 OpenAI api 格式，可供 agent 直接读取

```
        def process_requests():
        global fpga_connection

        while True:
            try:
                queue_item = request_queue.get()

                if len(queue_item) == 2:
                    _, data = queue_item
                    response_callback = None
                elif len(queue_item) == 3:
                    _, data, response_callback = queue_item
                else:
                    print(f"[H100] 错误: 无效的队列项格式: {queue_item}")
                    continue


                with connection_lock:
                    if fpga_connection is None or fpga_connection.conn is
```

```
            None:
            print("[H100]␣错误：未连接上位机")
            continue


    try:
        with connection_lock:
            current_conn = fpga_connection

        send_data = {
            "model": data.get("model"),
            "messages": data.get("messages"),
            "temperature": data.get("temperature"),
            "max_tokens": data.get("max_tokens")
        }

        response = current_conn.send_message(json.dumps(send_data
            ))


        openai_response = {
            "id": f"chatcmpl-{int(time.time())}",
            "object": "chat.completion",
            "created": int(time.time()),
            "model": openai_model,
            "choices": [{
                "message": {
                    "role": "assistant",
                    "content": response
                },
                "finish_reason": "stop",
                "index": 0
            }]
        }


        if response_callback:
            response_callback(openai_response)

    except Exception as e:
        print(f"[H100]␣处理请求失败:␣{str(e)}")

        with connection_lock:
```

```
                if fpga_connection:
                    fpga_connection.close()
                    fpga_connection = None

        except Exception as e:
            print(f"[H100]␣请求处理错误:␣{str(e)}")
            time.sleep(1)
```

- 与 agent 或网页进行收发消息交互

```
    def openai_endpoint():

    try:

        with connection_lock:
            if fpga_connection is None or fpga_connection.conn is None:
                return jsonify({"error": "未连接上位机"}), 503


        request_data = request.json
        print(f"[H100]␣收到Agent请求:␣{json.dumps(request_data,␣indent
            =2)}")

        response_event = threading.Event()
        response_data = [None]

        def response_callback(resp):
            response_data[0] = resp
            response_event.set()

        processing_data = {
            "messages": request_data['messages'],
            "model": request_data.get('model', 'glm'),
            "max_tokens": request_data.get('max_tokens', 1024),
            "temperature": request_data.get('temperature', 0.7),
            "response_callback": response_callback
        }

        request_queue.put((1, request_data, response_callback))
        print("发送给agent回复")
        #request_queue.put(("fpga", processing_data))
        response_event.wait(timeout=60)
```

```
        if response_data[0] is None:
            return jsonify({"error": "处理超时"}), 504

        return jsonify(response_data[0])

    except Exception as e:
        print(f"[H100] API错误: {str(e)}")
        return jsonify({"error": "服务器错误"}), 500
```

- 上位机监听服务器发送的消息，编译成 FPGA 可读的形式传给 FPGA

```
def other_server_communication():
    global other_server_socket
    server_socket = socket.socket(socket.AF_INET, socket.
        SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR
        , 1)
    server_socket.bind(("0.0.0.0", OTHER_SERVER_PORT)) # 监听所有接
        口
    server_socket.listen(1)
    print(f"上位机已在端口 {OTHER_SERVER_PORT} 监听H100连接...")
    while True:
        try:
            client_sock, addr = server_socket.accept()
            print(f"接受来自H100的连接: {addr}")
            other_server_socket = client_sock
            while True:

                length_data = other_server_socket.recv(4)
                if not length_data:
                    raise ConnectionError("连接已关闭")

                    # 解析消息长度
                msg_length = struct.unpack('>I', length_data)[0]

                chunks = []
                bytes_received = 0
                while bytes_received < msg_length:
                    chunk = other_server_socket.recv(min(msg_length -
                        bytes_received, 4096))
                    if not chunk:
                        raise ConnectionError("连接中断")
                    chunks.append(chunk)
```

```
                    bytes_received += len(chunk)

                full_message = b''.join(chunks).decode('utf-8')

                if full_message.endswith(END_MARKER.decode('utf-8')):
                    full_message = full_message[:-len(END_MARKER)]

                message_queue.put(full_message)
                print(f"收到完整服务器消息 (长度: {msg_length}): {
                    full_message[:50]}...")

        except (ConnectionError, socket.error) as e:
            print(f"服务器连接错误: {str(e)}")
            if other_server_socket:
                other_server_socket.close()
                other_server_socket = None
            time.sleep(5) # 等待后重连

        except Exception as e:
            print(f"处理服务器消息出错: {str(e)}")
            if other_server_socket:
                other_server_socket.close()
                other_server_socket = None
            time.sleep(1)

def main_kvcache_show(tokenizer, client, memory=0):
    global other_server_socket

    server_thread = threading.Thread(target=
        other_server_communication, daemon=True)
    server_thread.start()

    round = 1
    while True:

        query, is_server_message = collect_input()

        if query is None:
            time.sleep(0.1)
            continue

        try:
```

```
        data = json.loads(query)

        user_messages = [msg for msg in data.get("messages", [])
                         if msg.get("role") == "user"]

        if user_messages:

            actual_query = user_messages[-1]["content"]
        else:

            system_messages = [msg for msg in data.get("messages
                ", [])
                               if msg.get("role") == "system"]
            actual_query = system_messages[0]["content"] if
                system_messages else "没有提供查询内容"


        temperature = data.get("temperature", [])
        max_tokens = data.get("max_tokens", [])

        print(f"解析后的查询内容: {actual_query}")
        print(f"参数: temperature={temperature}, max_tokens={
            max_tokens}")

    except json.JSONDecodeError:
        print("无法解析JSON, 使用原始输入")
        actual_query = query
    except Exception as e:
        print(f"解析输入时出错: {str(e)}")
        actual_query = query

    prompt = "[Round {}]\n\n问: {}\n\n答: ".format(round,
        actual_query)
    print(f"构造的prompt: {prompt[:100]}...")
    inputs = tokenizer([prompt], return_tensors="pt")

    print("FPGA: ", end="")
    run_model_kvcache_show(client, inputs["input_ids"], inputs["
        input_ids"].shape[1], 0, memory)

    full_response = ""
    while True:
        state, ids_len, ids = get_next_ids(client)
```

```
            if state == 0:
                print("")
                main_time_show(client)

                if other_server_socket:
                    send_to_other_server(full_response)
                break

            generated_text = tokenizer.decode(ids)
            print(generated_text, end="", flush=True)
            full_response += generated_text

        round += memory
```

3. 开发计划表

- 未来会继续向现有的工具链中添加新的功能
- 除了在各个环节加入新的功能外，还将额外开发 Cli 命令行工具以便在服务器终端操作调用
- 苏灿同学会同时参与量化部分的 llmc 开发

子项目	开发者	已完成内容	6月29日	未来开发
量化	高琦	GPTQ	AWQ	VIT量化工具适配
打分		Lm-Evaluation-Harness EvalPlus	OpenCompass	llmc
GPU部署		vLLM	暂无	
权重处理	苏灿	Compiler-VCU128	多模态模型权重处理和上板	
API服务		Fast API	暂无	
调度器	高琦	GPU, Port	已完成	
WebUI	黎睿正	基础功能	适配后端GPU部署测试功能	完成WebUI开发内容
Cli	高琦	暂不开发	适配完毕已有功能	

图 5 开发计划表