南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

软件工程测试报告

车载多模态智能交互系统
测试报告

年级：2022 级

专业：计算机科学与技术

指导教师：李起成

林逸典　唐显达　姜宇　郭笑语
刘芳宜　王禹衡　何畅　董珺

2025 年 6 月 17 日

目录

一、 引言

我们团队精心研发了一款车载多模态智能交互系统软件，旨在为驾驶员和乘客提供更加便捷、安全和人性化的交互体验。该系统集成物理模态、语音模态和视觉模态（视线方向、手势动作和头部姿态）等多种模态，实现了播放音乐、地图导航和车辆状态监测等常用应用功能，还具备个性化配置、权限管理、日志记录等系统功能。

本测试报告将详细地记录本轮测试的环境、方法和流程。针对测试中发现的问题，将深入剖析并进行修复。测试涵盖功能测试、性能测试和安全性测试。功能测试重点检查系统各项功能是否符合设计要求、能否准确实现预期功能；性能测试评估系统在不同负载条件下的响应速度和处理能力，确保其高效稳定运行；安全性测试旨在发现系统可能存在的安全隐患，保障用户数据和系统运行安全。

在功能测试中，采用黑盒测试与白盒测试相结合的方法设计单元测试样例。黑盒测试从用户角度出发，不关注系统内部实现细节，仅验证输入与输出结果是否符合需求；白盒测试将重点关注“典型场景展示: 启动系统时自动开启应用功能”，检查该场景下能否准确识别用户意图、正确执行应用功能和系统功能。测试完成后，将给出详细的缺陷报告，记录缺陷的发现时间、位置和严重程度等信息，并对缺陷进行修复。

综上所述，我们对系统进行了全面测试，特此给出本测试报告。

二、 测试环境

(一) 软件环境

组件	配置说明
操作系统	Windows 11
Python 版本	Python 3.12
包管理工具	Conda + Pip
核心框架	PyTorch 2.7.0
音频处理	pyaudio，soundfile，webrtcvad
语音识别	funasr
图像与姿态识别	mediapipe，opencv-contrib-python
自然语言处理	spacy，pypinyin
辅助工具	numpy，scikit-learn，pygame
前端服务	基于 Flask 框架的网页服务

(二) 硬件环境

组件	配置说明
CPU	Intel(R) Core(TM) i7-14700HX 2.10 GHz
GPU	NVIDIA 显卡 4070
麦克风阵列	Realtek(R) Audio
摄像头	Integrated Camera(驱动程序提供商:Realtek)

三、 功能测试

(一) 黑盒测试

模块（被测点）	输入（模拟）	行为
初始化流程	–	记录日志，更新 UI，播报提示语音
视线判断	视线方向持续为“中间”超过 3 秒	跳出循环，进入下一流程
视线判断	视线方向长时间为非“中间”	保持循环
语音确认	正确识别语音 + 驾驶员身份一致	执行对应功能调度
语音确认	拒绝指令 + 驾驶员身份一致	跳过该功能执行
语音确认	非驾驶员说话	无动作发生
手势确认	手势为“竖起大拇指”	执行功能调度
手势确认	手势为“摇手”	拒绝功能执行
头部动作	姿态为“点头”	同意功能执行
头部动作	姿态为“摇头”	拒绝功能执行
功能调度	三轮模态同意输入	调用应用功能三次
功能拒绝	三轮模态拒绝输入	无应用功能被调度
流程结束	–	正常退出
异常分支	模态信息缺失	持续等待输入

表 1: “典型场景展示: 启动系统时自动开启应用功能”测试用例表

表1 展示了针对“典型场景展示: 启动系统时自动开启应用功能”所设计的黑盒测试用例。测试覆盖了系统初始化、模态判断、用户确认与功能调度等各个环节，模拟了用户在交互过程中的多种典型操作情境。

测试用例分别考虑了视线方向保持或未保持、语音输入内容与发言者身份是否匹配、手势识别结果与意图的一致性，以及头部动作的正反表达等输入情境，以验证系统是否能正确理解用户意图并进行响应。此外，测试还设计了连续三轮功能确认场景，并区分了“全部同意”与“全部拒绝”的输入路径，以保证系统在多轮交互中功能执行的完整性与可控性。

异常输入方面，测试模拟了模态信息缺失或无响应等边界情况，以验证系统是否具备容错性和稳态等待能力，不会因模态异常导致程序崩溃或逻辑错误。

模块（功能类型）	输入（模拟）	行为
music_getlist	空参数	返回音乐列表
music_play	空或音乐名参数	播放指定或默认音乐
music_pause	–	暂停当前音乐
music_unpause	–	恢复播放音乐
music_change_pause	–	切换播放/暂停状态
navigation	–	触发导航功能
monitor_getlist	–	获取车辆状态监测数据
monitor_jump	–	跳转导航页面
enter	–	典型场景展示
get_application_names	–	返回用户可选的功能名称列表
to_string（有效输入）	枚举类 Application.type.xx	返回对应中文描述名称
to_string（无效输入）	无法匹配的 Enum 值	返回“未知功能”

表 2: 应用功能模块测试用例设计表

表2 展示了针对应用功能模块的功能测试用例设计，采用黑盒测试方法，旨在验证各类功能接口在不同输入条件下的行为是否符合预期。该模块涵盖音乐播放、地图导航和车辆状态监测等多个典型功能，测试用例围绕“输入—行为”双向映射构建，确保系统外部调用接口的完整性和鲁棒性。

测试覆盖三类功能接口：其一是具备参数输入的可变响应函数，其二是直接触发动作的无参函数，其三为基础信息查询类接口。此外，测试还特别设计了边界输入情形，如枚举转换失败场景，以检测系统在处理无效输入时的健壮性与容错性。

通过该表可以确认，Application 模块的核心调度逻辑对每一项功能类型均提供了明确可控的入口，外部系统可在不依赖其内部实现逻辑的前提下，验证功能是否调用成功、是否返回有效数据或执行目标操作。

模块（方法名）	输入（模拟）	行为
getlist()	音乐目录包含若干音频文件	返回文件名列表
find()	输入关键词与某个文件名部分匹配	返回该文件路径
find()	输入关键词与所有文件名不匹配	返回空字符串
play()	默认参数或输入关键词正确	调用 jump_to_page("music")，加载并播放音乐
pause()	正在播放音乐	暂停音乐播放，paused 设为 True
pause()	没有播放音乐	不执行暂停逻辑，paused 设为 False
unpause()	当前已暂停	恢复播放，paused 设为 False
unpause()	当前未暂停	无操作
change_pause()	当前为播放状态	自动暂停
change_pause()	当前为暂停状态	自动恢复播放

表 3: Music 模块功能测试用例设计表

表3 展示了 Music 模块的功能测试用例，验证了音乐播放控制模块在不同输入下的行为响应是否符合用户期望。测试覆盖了典型的模块接口函数，包括音乐文件的读取和播放控制。

测试用例重点关注模块在“播放状态”和“暂停状态”两种上下文中的状态切换机制，确保音乐播放逻辑的可控性和状态一致性。例如，pause() 和 unpause() 的测试场景明确了模块对于“当前状态”的响应策略，避免冗余操作或异常行为；find() 接口也针对路径匹配失败进行了异常处理测试，以保证其鲁棒性。

模块（方法名）	输入（模拟）	行为
navigate()	from_ 和 to_ 均为 None	调用 navigate()（无参数），并跳转至导航界面
navigate()	提供 from_ 和 to_（字符串）	调用 navigate(from_, to_)，并跳转至导航界面

表 4: Navigation 模块功能测试用例设计表

表4 展示了 Navigation 模块的功能测试用例，聚焦于 navigate() 方法在不同输入条件下的行为差异。测试用例验证了该方法是否能够正确调用导航功能并完成页面跳转。

测试涵盖两类典型输入：一类为未提供任何起止地点参数，系统应调用无参的 navigate() 接口并跳转至默认导航界面；另一类为显式提供起点和终点信息，系统应基于输入参数调用对应的导航函数，并跳转至界面以展现路径信息。

模块（方法名）	输入（模拟）	行为
`init_gazer()`	正常摄像头输入；设置 `--debug` 参数	视线跟踪器成功注册并启动; 打印调试信息并朗读提示语
`init_speecher()`	禁用唤醒词（`--no-wake`）、注册新用户（`--register-name test`）	正确初始化语音模块, 关闭唤醒词功能，并注册“test”声纹
`init_headpose()`	参数 `--method gru` 或 `geom`；模拟无摄像头异常	初始化对应识别器; 摄像头异常时终止并报错
`init_static_gesture()`	模型文件路径缺失或无效	检查路径后报错退出; 不进行初始化
`control()`	模拟 `get_all_key_info()` 返回各模态输入	能正确解析语音、手势输入并交由 individuation 模块处理; 未知模态时报错

表 5: 多模态控制模块测试用例设计

表5 针对多模态控制模块的核心方法进行了功能测试设计，覆盖了四个初始化流程与主控制逻辑。

对于 `init_gazer()`、`init_speecher()`、`init_headpose()` 和 `init_static_gesture()` 等初始化函数，测试用例关注其在输入参数不同（如 `--debug`、声纹注册开关、模型路径是否存在等）情况下的系统行为，如模态注册是否成功、摄像头异常是否能正确报错退出等。这些均为模块可观测的外部行为，符合黑盒测试关注输入输出结果的特性。

此外，`control()` 方法的测试用例验证系统在接收来自不同模态的模拟输入时，是否能按预期调用个性化模块处理信息，并对未知模态输入给出明确错误提示。测试强调了模块对典型使用场景的覆盖性和异常输入的健壮性，确保其整体交互调度逻辑稳定可靠。

(二) 白盒测试

我们基于“典型场景展示: 启动系统时自动开启应用功能”的控制流图设计测试用例，对典型场景进行白盒测试。该流程包含从系统初始化到用户完成多轮功能确认的完整逻辑，涉及语音识别、手势识别和头部姿态识别等多个模态的融合与判断。

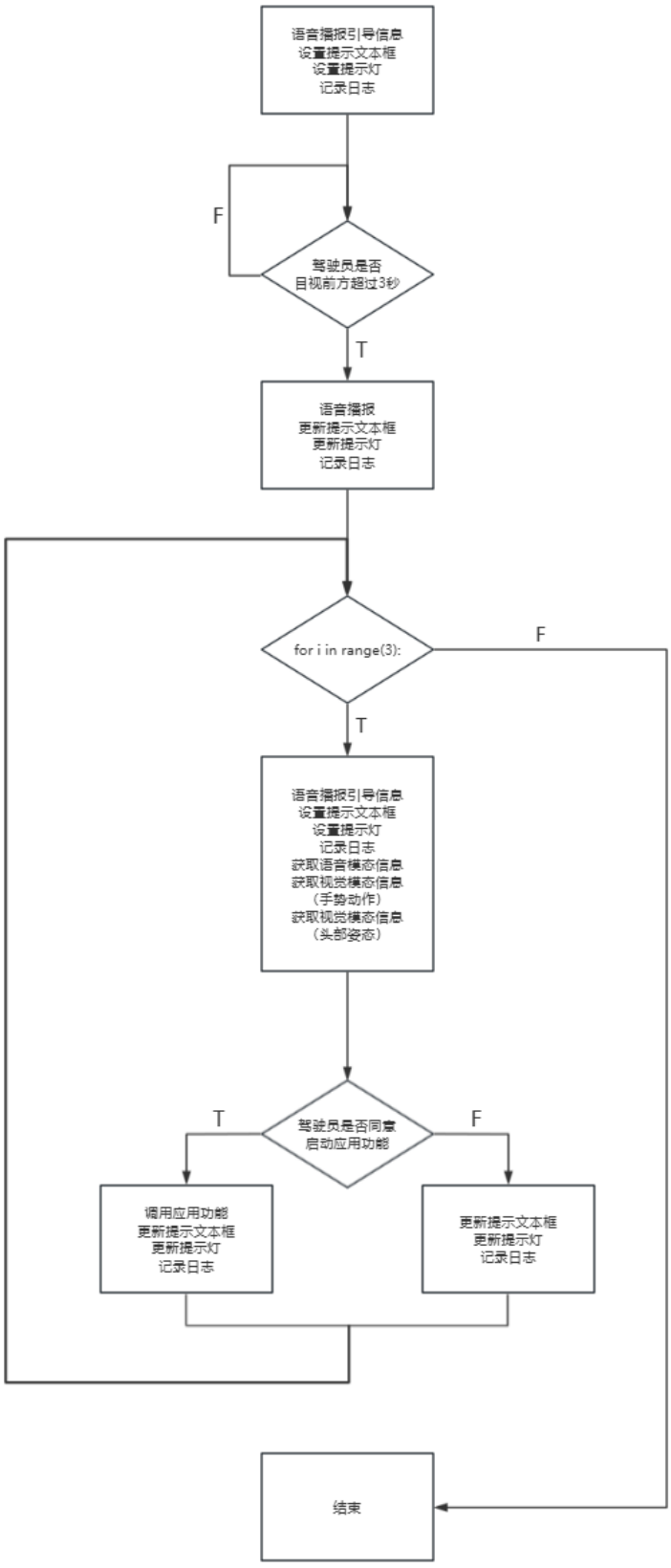

图 1: Enter 流程图

在测试设计中，我们采用了语句覆盖、分支覆盖和路径覆盖三种白盒测试标准，对每一个控

制流判断点、分支出口及功能路径进行了全面覆盖。具体而言，我们测试了用户视线保持不足与保持超过三秒两种情况，以验证流程是否能在符合条件时准确进入功能确认阶段。随后，分别语音输入为“同意”和“拒绝”，并结合驾驶员身份进行判断，确保仅在语音身份匹配的情况下才能触发操作。同样，我们也验证了手势输入（如竖起大拇指、摇手）和头部姿态（如点头、摇头）对系统反应路径的影响。

为了确保系统的健壮性，我们还设计了输入缺失的场景。例如，当用户不进行任何输入或模态识别失败时，系统应能保持等待状态而不崩溃或进入错误路径。同时，我们验证了三轮典型功能交互（如导航、车辆状态监测、音乐播放）均能在多模态“同意”或“拒绝”输入下正确完成功能调度或跳过。

从执行结果看，本轮白盒测试已覆盖了流程图中所有主流程路径及关键分支逻辑，所有语句和判断分支均至少被触发一次。流程图中的循环、条件跳转与多模态输入路径均能通过已设计用例进行验证，控制逻辑闭环明确，系统在典型使用场景下表现出良好的功能完整性与稳定性。

(三) 缺陷报告

缺陷编号	001
缺陷内容	系统响应了非驾驶员发出的语音指令，存在安全风险。
复现步骤	启动系统后，由非驾驶员发出“同意导航”等语音指令。
预期结果	系统应校验说话人身份，仅响应注册驾驶员的语音输入。
实际结果	系统直接执行了非驾驶员的语音指令。
严重程度	高
提交人	何畅
修复情况	已修复

缺陷编号	002
缺陷内容	多条语音播报重叠，语音输出卡顿，听感不清晰。
复现步骤	系统连续播报多个推荐提示，如“是否为您监测”、“是否为您导航”等。
预期结果	播报之间有间隔，每条语音清晰完成后再播放下一条。
实际结果	多条语音合成任务重叠播放，导致卡顿和混音。
严重程度	中
提交人	何畅
修复情况	已修复

缺陷编号	003
缺陷内容	提示文本重复刷新，界面频繁闪烁，影响用户体验。
复现步骤	系统进入推荐流程时频繁调用 `update_note(text)`。
预期结果	每一条提示只更新一次，界面稳定显示提示内容。
实际结果	同一提示被多次刷新，导致界面抖动和闪烁。
严重程度	中
提交人	何畅
修复情况	已修复

四、 性能测试

(一) 典型场景展示: 启动系统时自动开启应用功能

函数位置	调用次数	总时间 (s)	累计时间 (s)	说明
sounddevice.py:381(wait)	4	0.000	16.227	语音播报阻塞等待
threading.py:637(wait)	4	4.298	15.999	音频线程等待
threading.py:323(wait)	4	0.000	11.687	等待锁释放
{_thread.lock}.acquire	21/17	11.647	11.686	线程锁竞争
time.sleep	9	9.003	9.003	等待用户输入
serialization.py:2060(persistent_load)	52/9	2.028	6.462	模型权重加载
sounddevice.py:1155(close)	7	0.228	4.541	关闭音频流
torch.nn.modules.module:1747(_wrapped_call_impl)	2368/4	0.003	2.153	推理调用包装器
kokoro.model.py:121(forward)	4	0.010	2.153	模型前向推理
kokoro.istftnet.py:407(forward)	4	0.000	1.300	音频解码

表 6: 典型场景运行期间函数调用性能分析（Top 10 按累计时间排序）

在对典型场景进行 `cProfile` 分析后，我们得出如表 6 所示的前 10 项函数调用开销统计。从表中可以看出，语音相关函数 `sounddevice.wait()` 占据了最多的累计时间（达 16.2 秒），约占总运行时间的 45%，成为性能瓶颈的主要来源。此外，`threading.wait` 和 `time.sleep` 等阻塞性调用也带来显著开销。

尽管我们已对音频处理流程进行了多项优化，例如引入异步语音播放机制以降低系统延迟，但模型推理部分（如 `kokoro.model.forward`）仍累计耗时超过 2 秒，占总时长的 6%，表明语音合成/识别模块在多模态交互中的负载仍不可忽视。未来我们计划进一步精简语音模块的计算路径，并探索更高效的并发调度策略，以进一步提升整体响应速度。

(二) 多模态控制模块

函数位置	调用次数	总时间 (s)	累计时间 (s)	说明
torch.serialization:persistent_load	76/7	4.449	23.531	加载模型参数
tensorflow._pywrap_tfe.TFE_Py_FastPathExecute	18629	13.334	13.785	TensorFlow 模型推理
time.sleep	36896/803	81.137	9.595	等待用户响应
pyaudio.read	3275/152	0.549	6.782	从麦克风读取音频流
requests.sessions:request	16	0.000	4.588	请求远程服务
requests.adapters:send	16	0.000	4.267	HTTP 数据发送
sounddevice:wait	1	0.000	3.064	语音播放阻塞
requests.api:get	5	0.000	2.854	请求更新信息
tensorflow.ops:_create_graph_constant	10429	0.025	2.664	TensorFlow 图构建
socket:readinto	72	0.000	2.577	网络请求读取

表 7: 多模态控制模块运行期间函数调用性能分析（Top 10 按累计时间排序）

从对多模态控制模块进行 `cProfile` 性能分析的结果（见表 7）可见，该模块的主要开销集中在以下几个方面:

首先，模型参数的加载函数 `torch.serialization.persistent_load` 累计耗时达 23.5 秒，反映出系统在初始化过程中模型反序列化操作的高开销；其次，TensorFlow 后端的推理执行函数 `TFE_Py_FastPathExecute` 累计耗时超过 13 秒，是系统运行期间的另一个关键瓶颈。

此外，`time.sleep` 函数（累计约 9.6 秒）和 `pyaudio.read`（6.8 秒）显示系统在交互等待过程中存在显著的阻塞开销。语音合成使用的 `sounddevice.wait` 也耗费了 3 秒左右的运行时

间，表明同步语音播放依旧是交互延迟的重要来源。

尽管我们已对部分网络请求引入了缓存和超时控制,但系统中如 `requests.send` 和 `socket.readinto` 等操作的累计耗时仍接近 5 秒，可能与模型更新检查、远程语音识别或配置加载有关。为进一步提升响应效率，我们计划引入本地缓存策略、减少远程依赖，并优化请求调度机制，以降低网络延迟带来的影响。

(三) 摄像头管理器

在车载多模态智能交互系统里，我们采用全局唯一的摄像头管理器来统一管理摄像头资源。该管理器借助 cv2.VideoCapture 获取摄像头资源，并支持头部姿态子系统、手势动作子系统、视线方向子系统以及 UI 界面的摄像头区域同步读取摄像头数据，以此规避多种视觉模态直接竞争摄像头资源所引发的性能降低乃至程序崩溃问题。

五、 安全性测试

(一) 异常处理

在系统运行的全流程中，我们合理地运用 try-catch 语句，精准且全面地捕获各类可能抛出的异常错误，同时依据错误的性质与严重程度，制定并实施恰当有效的处理方案，以此保障系统能够长期、持续且稳定地运行。

(二) 权限系统

为验证权限系统在驾驶员身份校验方面的安全性，我们构造了一个模拟攻击测试用例，使用声纹识别模块输入非法用户（攻击者 attacker）所发出的合法语音指令“同意导航”，以此测试系统是否会被非授权用户误触发任务调度功能。

系统在运行过程中正确识别出说话人并将其标记为非法用户，即使语音指令语义正确，由于攻击者身份不匹配当前设置的驾驶员，系统未将其作为有效意图处理，并予以警告。测试过程中，系统持续处于等待合法驾驶员确认的状态，并未触发任何调度动作。

六、 总结

综上所述，在本测试报告中，我们针对车载多模态智能交互系统展开了全面的测试工作。通过功能测试确认了系统各项功能符合设计要求、能够准确实现预期功能；性能测试评估了系统在不同负载条件下的响应速度和处理能力，确保系统地高效稳定运行；安全性测试对系统可能存在的安全隐患进行了排查，切实保障用户数据和系统运行安全。测试期间，针对发现的问题，我们进行了详尽记录与深入分析，并采取针对性的修复措施，确保系统质量得到有效提升。