国芯微
NationalChip

人工智能实验室

Alchemy 中文使用文档

文档版本 V3.6.2

发布日期 2026-03-12

免责声明与版权公告

本文档包含的信息（包括网页参考）如有更改，恕不另行通知。

本文档中所有第三方的信息均“按原样”提供，对其真实性和准确性不作任何保证。

本文不作任何明示或默示的担保，包括但不限于适销性、不侵犯知识产权、适用于特定目的的担保，也不对任何提案、规格或示例中可能产生的任何其他担保作出承诺。对于使用本文档信息所引起的任何责任（包括侵犯任何所有权的责任），均予以免责。本文档未以任何形式（无论是禁止反言或其他方式）授予任何知识产权方面的明示或默示许可。

商标声明

本文档中提及或涉及的 、NationalChip 商标均为 NationalChip 的商标。

任何单位或个人未经 NationalChip 事先书面许可，不得使用此类商标。

本文档提及的所有商品名称、商标及注册商标，其所有权均归各自所有者，特此声明。

重要提示

1. 所购买的产品/服务以最终签署的商业合同为准。文档内容可能超出您的购买范围。
2. 除非合同中有明确规定，本文档不作任何明示或默示的保证。
3. 文档更新可能不会另行通知。本文档仅供参考之用，不作任何担保承诺。

如需技术咨询或与应用相关的问题，请就近联系我们的销售网点。

版本历史

版本	修订日期	修订说明
V1.0	2023-04-17	初始发布版本
V2.0	2025-05-14	更新框架，支持一键run
V3.1.0	2025-01-20	更新框架，支持自动数据合成、自动上传、一键训练
V3.2.0	2025-07-17	优化自动数据合成方法，优化自动上传，增加model_k
V3.2.1	2025-08-12	修复安装路径问题，优化corpus配置
V3.3.1	2025-08-14	支持train-agent，支持一键训练流程
V3.4	2025-08-31	支持model-t，增加故障排除新章节，优化自动配置
V3.5	2025-09-12	支持多通道输入，支持model-f，优化算法
V3.6.1	2026-01-13	支持GX8008C编译部署,修复一键发布的问题
V3.6.2	2026-03-12	更新模型与适配芯片选项,支持npu编译器1.6.5

Contents

List of Figures

List of Tables

1 Alchemy 系统简介

1.1 关于 Alchemy

Alchemy 是杭州国芯微电子基于 TensorFlow 框架开发的一套以训练稳定 KWS（关键词识别）模型的微调与部署系统，可让用户在特定应用场景下快速获得可用的 KWS 模型，并完成从数据到部署的一站式自动化流程。

1.2 主要功能

- 自动数据处理：支持 TTS 合成，支持音频数据的自动提取、预处理
- 模型微调：提供多种模型结构和训练策略，通过微调获得目标模型
- 训练过程可视化：支持训练过程可视化、损失变化记录；自动合成测试集、自动测试并导出 Excel 报告

1.3 模型与适配芯片

表 1-1：模型与适配芯片选择

模型型号	模型大小/参数量	语种	芯片型号	是否支持低功耗
model_b	150KB/100K	中文	GX8002	支持（不推荐）
model_g	450KB/350K	中文	GX8002D/GX8008C	不支持 / model_t+model_g 支持（可用于大小模型）
model_k	550KB/450K	中文	GX8302B	不支持
model_f	550KB/450K	中文	GX8003/GX8005/GX8-006/GX8008C	不支持
model_t	120KB/90K	中文	GX8002D	支持（仅用于大小模型）

1.4 目标用户

算法工程师、产品工程师

1.5 产品特色

一站式自动化流程：数据处理 → 数据准备 → 模型训练 → NPU 编译文件生成 → 模型部署

2 Alchemy 系统安装与配置

2.1 系统的硬件与软件要求

2.1.1 硬件要求

- CPU: 8 核以上
- 内存: 128GB 以上 + 48G 以上虚拟内存
- 存储: 4T 以上可用空间 (建议 SSD)
- GPU: 4090Ti

2.1.2 软件要求

- 操作系统: Ubuntu 18.04/20.04 LTS
- NVIDIA 驱动: 550.90.07 或更高版本
- 网络: 能访问 FTP 服务器

2.1.3 推荐配置环境（示例）

- 4090Ti：NVIDIA-SMI 525.105.17，CUDA 12.0

注意：驱动版本需与系统、显卡和 CUDA 版本匹配，请以实际环境为准。

2.2 推荐使用流程

2.2.1 新客户（首次安装）

1) 文件下载：运行 download_alchemy_files.sh 下载所有必要文件
2) 环境安装：运行 extract_alchemy_files.sh 自动解压和安装环境
3) 环境配置：运行 auto_setup_environment.sh 配置系统环境变量和依赖
4) 路径配置：运行 corpus_data_setting.sh 配置 alchemy 工程路径参数
5) 环境验证：先运行 example_local 验证环境
6) 正式开发：使用 example_0 进行项目开发

2.2.2 V3.0+ 升级客户（已有环境）

1) 下载新版本：仅下载 alchemy.tar.gz
2) 备份旧版本：备份现有 alchemy 目录（可选）
3) 解压新版本：解压到现有 workspace 目录
4) 工程路径配置：运行 corpus_data_setting.sh 更新路径参数
5) 环境验证：运行 example_local 验证升级是否成功
6) 正式开发：继续使用 example_0 进行项目开发

2.3 全新安装与手动安装的细节

2.3.1 场景一：全新安装（新客户）

适用客户：第一次使用 Alchemy，没有任何环境

- 方式一：自动化安装（推荐）

完整自动化流程：

```
# 1. 下载所有必要文件
bash download_alchemy_files.sh

# 2. 自动解压和安装环境
bash extract_alchemy_files.sh

# 3. 配置系统环境变量和依赖
cd ~/workspace/corpus
bash auto_setup_environment.sh

# 4. 配置 alchemy 工程路径参数
bash corpus_data_setting.sh

# 5. 重新加载环境
source ~/.bashrc

# 6. 验证环境
cd ~/workspace/alchemy/example_local
bash run_simple.sh
```

- 方式二：手动安装

适用场景：自动化安装失败或需要自定义安装过程

步骤 1：手动下载文件

```
# 创建下载目录
mkdir -p ~/downloads
cd ~/downloads

# 下载所有必要文件
wget --ftp-user=test --ftp-password=test@1234 ftp://ftp.nationalchip.com/alchemy/V3.6/alchemy.tar.gz
wget --ftp-user=test --ftp-password=test@1234 ftp://ftp.nationalchip.com/alchemy/V3.6/corpus.tar.gz
wget --ftp-user=test --ftp-password=test@1234 ftp://ftp.nationalchip.com/alchemy/V3.6/Miniforge3-Linux-x86_64.
```

(续下页)

(接上页)

```
↪sh
wget --ftp-user=test --ftp-password=test@1234 ftp://ftp.nationalchip.com/alchemy/V3.6/py36_tf115.tar.gz
wget --ftp-user=test --ftp-password=test@1234 ftp://ftp.nationalchip.com/alchemy/V3.6/SenseVoice_py311.tar.gz
wget --ftp-user=test --ftp-password=test@1234 ftp://ftp.nationalchip.com/alchemy/V3.6/cosyvoice.tar.gz
wget --ftp-user=test --ftp-password=test@1234 ftp://ftp.nationalchip.com/alchemy/V3.6/th_12.tar.gz
```

步骤 2：检查 NVIDIA 驱动

```
# 检查是否已安装 NVIDIA 驱动
nvidia-smi -L | cat

# 如果显示以下信息，说明没有安装驱动
# Command 'nvidia-smi' not found, but can be installed with:
# apt install nvidia-340          # version 340.108-0ubuntu5.20.04.2, or
# apt install nvidia-utils-390    # version 390.157-0ubuntu0.20.04.1
# ... 其他版本选项
```

安装 NVIDIA 驱动（如需要）：

```
# 禁用默认驱动
sudo vi /etc/modprobe.d/blacklist.conf  # 添加 blacklist nouveau 到该文件中
sudo update-initramfs -u
# 重启目标系统
sudo reboot

# 安装 NVIDIA 显卡驱动
sudo ./NVIDIA-Linux-x86_64-550.90.07.run  -no-x-check -no-nouveau-check -no-opengl-files
# 重启目标系统
sudo reboot
```

步骤 3：解压 Alchemy 项目

```
# 1. 解压主工程文件
mkdir -p ~/workspace
tar -zxvf alchemy.tar.gz -C ~/workspace

# 2. 解压语料库
tar -xzvf corpus.tar.gz -C ~/workspace  # 解压到 corpus 文件夹
cd ~/workspace/corpus
bash corpus_data_setting.sh    ### 运行语料库迁移脚本
bash auto_setup_environment.sh ### 运行环境配置

# 3. 手动配置环境变量（如果自动配置失败）
# 将以下内容添加到 ~/.bashrc 文件中，并将 `pub` 改为您的目标系统 pub
export PYTHONPATH=/home/pub/corpus/tts/VOICE/icefall/../icefall:$PYTHONPATH
export PYTHONPATH=/home/pub/corpus/tts/VOICE/valle/../valle:$PYTHONPATH
export PYTHONPATH=/home/pub/corpus/tts/VOICE/k2/../k2:$PYTHONPATH
# 配置完成后生效
source ~/.bashrc
```

步骤 4：安装 Miniforge3 和 Python 环境

```
# 1. 安装 miniforge3
./Miniforge3-Linux-x86_64.sh  ## 安装时根据提示确认安装路径，输入 y 确认
source ~/.bashrc

# 2. 解压 Python 环境
tar xzvf py36_tf115.tar.gz -C ~/miniforge3/envs
tar xzvf th_12.tar.gz -C ~/miniforge3/envs
tar xzvf SenseVoice_py311.tar.gz -C ~/miniforge3/envs
tar xzvf cosyvoice.tar.gz -C ~/miniforge3/envs

# 3. 配置 miniforge3 环境变量
# 将以下内容添加到 ~/.bashrc 文件中，并将 `pub` 改为您的目标系统 pub
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/home/pub/miniforge3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/home/pub/miniforge3/etc/profile.d/conda.sh" ]; then
        . "/home/pub/miniforge3/etc/profile.d/conda.sh"
    else
        export PATH="/home/pub/miniforge3/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<
# 配置完成后生效
source ~/.bashrc
```

步骤 5：配置 Torch 模型缓存及路径替换

```
# 1. 配置模型缓存
mkdir -p ~/.cache/
cp -r ~/workspace/corpus/cache/* ~/.cache/

# 2. 路径替换
sed -i "s|/home/sundy|$HOME|g" ~/miniforge3/envs/py36_tf115/bin/pip
sed -i "s|/home/sundy|$HOME|g" ~/miniforge3/envs/py36_tf115/bin/gxnpuc
sed -i "s|/home/sundy|$HOME|g" ~/miniforge3/bin/conda
```

步骤 6：安装依赖库

```
# 重新加载环境变量
source ~/.bashrc

# 激活 Python 环境
conda activate py36_tf115

# 安装 logfbank
cd ~/workspace/alchemy/tools/logfbank
python setup.py develop
```

(续下页)

(接上页)

```
# 安装 ffmpeg
sudo apt install ffmpeg

# 安装 npu-compiler
conda activate py36_tf115
pip install npu-compiler==1.6.5
```

步骤 7：验证安装

```
# 重新加载环境
source ~/.bashrc

# 验证环境
cd ~/workspace/alchemy/example_local
bash run.sh
```

2.3.2 场景二：升级安装（V3.0+ 客户）

适用客户：已安装过 V3.0 及以后但低于 V3.6 的版本，仅需更新代码

简化升级流程：

```
# 1. 下载新版本文件（仅下载 alchemy.tar.gz）
wget --ftp-user=test --ftp-password=test@1234 ftp://ftp.nationalchip.com/alchemy/V3.6/alchemy.tar.gz

# 2. 备份旧版本（可选）
mv ~/workspace/alchemy ~/workspace/alchemy_backup

# 3. 解压新版本
tar -zxvf alchemy.tar.gz -C ~/workspace


# 4. 配置路径参数（关键步骤）
cp ~/workspace/alchemy/tools/* ~/workspace/corpus ## 更新参数配置脚本
cd ~/workspace/corpus
bash corpus_data_setting.sh

# 5. 重新加载环境
source ~/.bashrc

# 6. 验证环境
cd ~/workspace/alchemy/example_local
bash run.sh
```

2.4 自动化脚本说明

1) download_alchemy_files.sh
   - 功能：从 FTP 服务器自动下载所有必要文件
   - 下载文件：alchemy.tar.gz、corpus.tar.gz、Miniforge3-Linux-x86_64.sh、py36_tf115.tar.gz、th_12.tar.gz、SenseVoice_py311.tar.gz、cosyvoice.tar.gz
2) extract_alchemy_files.sh
   - 功能：自动解压和安装环境
   - 执行：解压主工程至 ~/workspace/alchemy；解压语料库至 ~/workspace/corpus；安装 Miniforge3；解压 Python 环境至 ~/miniforge3/envs
3) auto_setup_environment.sh
   - 功能：系统配置环境变量和依赖
   - 执行：配置 conda 环境变量；安装 logfbank、npu-compiler 等基础依赖；更新 pip/conda 路径；设置模型缓存目录
4) corpus_data_setting.sh
   - 功能：在 alchemy 工程内部配置路径参数
   - 执行：检测当前用户路径；替换硬编码路径；更新 PYTHONPATH；处理 alchemy 配置文件路径

2.5 安装完成后的文件结构

```
~/workspace/
├── alchemy/                # 主工程目录
│   ├── alchemy/            # alchemy 系统代码
│   ├── example_local/      # 环境验证项目（5-10 分钟）
│   ├── example_0/          # 正式项目模板1
│   └── docs/V3.6/          # 文档目录
├── corpus/                 # 语料库和配置脚本
│   ├── download_alchemy_files.sh  # 一键下载脚本
│   ├── extract_alchemy_files.sh   # 一键解压安装脚本
│   ├── auto_setup_environment.sh  # 一键系统配置
│   └── corpus_data_setting.sh     # 一键参数配置
│   └── common_data/                # ASR数据和克隆源
│   └── tts/                # 克隆相关库
│   └── cache/              # 克隆相关模型
└── miniforge3/             # Python 环境
```

3 Alchemy 系统开发与实例

3.1 项目与示例概览

项目目录结构：

```
alchemy/
├── example_local        # 环境验证通用示例项目目录
├── example_0            # 用户实际项目训练项目目录
├── docs                 # 使用手册
└── alchemy              # 核心训练代码
```

表 3-1：项目选择指南

场景	目录	用途	时间	数据规模
环境验证	example_local	验证安装环境，快速体验	5-10 分钟	少量测试数据
正式项目	example_0	实际产品开发，完整训练	8-24 小时	完整训练数据库

3.2 快速验证示例（example_local）

步骤一：创建新项目目录

```
# 找到 home 路径目录: /home/pub
# 找到 alchemy 解压存储路径: /home/pub/workspace

# 复制示例项目模板
cd ~/workspace/alchemy
cp -r example_local user1_prj  # 复制环境验证为自定义示例项目 user1_prj 可自定义
```

步骤二：修改项目配置

需要修改的配置文件：top_conf.yaml 与 conf/prj_conf.yaml

1）修改 top_conf.yaml（项目基本配置）

```
# 进入项目目录（请将"pub"替换为您的实际 pub）
cd /home/pub/workspace/alchemy/user1_prj
vi top_conf.yaml
```

当前配置示例：

```
cmd_conf:
  main_cmd:
  - 小树你好
  - 你好小灯|你好灯光
  other_cmd:
  - 请你开灯|打开灯光
  - 关闭灯光
## 同一功能的指令词之间用|分隔
model_conf:
  chip: gx8002
  model_name: model_f
project_conf:
  gpu_id: 0
  project_name: example_f
  pub_path: ./output
  standard: 通用
  work_space: ./work_space
```

修改建议：

```
cmd_conf:
  main_cmd:
  - 小爱小爱          # 修改为您的主要指令词
  - 你好小爱|小爱同学  # 多个指令词用|分隔
  other_cmd:
  - 打开空调|打开风扇  # 修改为您的其他指令词
  - 关闭空调|关闭风扇  # 同一功能的指令词用|间隔
  - 调大音量|调小音量
## 注意：目前不支持英文指令词，指令词仅支持中文，不包含任何符号或阿拉伯数字
model_conf:
  chip: gx8002       # 根据您的芯片型号修改（gx8002/gx8008c）
  model_name: model_f # 根据您的需求修改（model_f/model_k/model_t/model_g等）
project_conf:
  gpu_id: 0          # 根据您的GPU编号修改
  project_name: my_project  # 修改为您的项目名称
  pub_path: ./output  # 模型输出路径，一般不需要修改
  standard: 通用      # 标准类型，一般不需要修改
  work_space: ./work_space  # 工作空间路径，一般不需要修改
```

2）修改 conf/prj_conf.yaml（数据路径配置）

```
vi conf/prj_conf.yaml
```

需要修改的关键部分：

```
database_conf:
  train_database_conf:
    hifi_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/hifi/train/list/asr'
    valle_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/valle/train/list/asr'
    cosyvoice_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/cosyvoice/train/list/asr
↪'
    # ... 其他配置保持不变
```

(续下页)

(接上页)

```
  test_database_conf:
    hifi_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/hifi/test/list/asr'
    valle_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/valle/test/list/asr'
    cosyvoice_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/cosyvoice/test/list/asr'
↪
    # ... 其他配置保持不变
```

3) 修改 conf/train_conf/*.yaml（训练配置文件）

```
vi conf/train_conf/v0.1.1.yaml
```

需要修改的路径：

```
dataset_conf:
  train_aug_conf:
    noise_dir: '/root/workspace/corpus/common_data/noise/add_noise'
    ground_noise_dir: '/root/workspace/corpus/common_data/noise/ground_noise'
    reverb_dir: '/root/workspace/corpus/common_data/RIR'
    white_noise_dir: '/root/workspace/corpus/common_data/noise/white_noise'
  domain_aug_conf:
    noise_dir: '/root/workspace/corpus/common_data/noise/add_noise'
    ground_noise_dir: '/root/workspace/corpus/common_data/noise/domain_noise'
    reverb_dir: '/root/workspace/corpus/common_data/RIR'
    white_noise_dir: '/root/workspace/corpus/common_data/noise/white_noise'
  common_tfr_conf:
    asr_tfr_dir: ['/root/workspace/corpus/common_data/ASR']
deploy_conf:
  neg_path: '/root/workspace/corpus/common_data/test_set/neg_8002'
```

重要提醒：

- 请将所有的 /root 替换为实际用户路径（如 /home/username 或 $HOME）
- 如使用自有数据，请将路径指向您生成的数据列表
- 其他配置参数一般不需要修改

步骤三：运行环境验证示例，训练报告生成

1) 运行训练脚本（一键完成：数据合成、数据准备、模型训练、模型测试和报告）

```
bash run.sh
```

2) 运行结果说明

- 约 10 分钟完成，生成 2 个模型与测试报告
- 输出路径：output/pub_model；日志：output/log；put.yaml 记录模型信息
- 项目工作路径：/home/pub/workspace/alchemy/user1_prj/test_project/project/self_project
- version 目录包含 v0.1.1、v0.2.1 等版本文件；测试报告路径示例：v0.1.1/result/ori/default/*excel；可测试模型路径：v0.1.1/result/ori/default/*default/model*.h

3.3 实际项目示例（example_0）

步骤一：创建新项目目录

```
cd ~/workspace/alchemy/
cp -r example_0 dryer_prj
cd dryer_prj
```

步骤二：修改数据存储路径（若未执行自动配置脚本）

```
# home: /home/pub
# alchemy: /home/pub/workspace/alchemy
# 通用数据路径: /home/pub/workspace/corpus
vim conf/prj_conf.yaml
```

步骤三：修改指令词、模型以及项目路径

```
vi /home/pub/workspace/alchemy/dry_prj/top_conf.yaml
vi /home/pub/workspace/alchemy/dry_prj/conf/prj_conf.yaml
```

配置修改说明：

- 参考上文“3.2 快速验证示例”的配置步骤
- 将 /home/pub 替换为实际用户路径
- 按需修改指令词、模型型号与芯片型号
- 还需修改 conf/train_conf/*.yaml 中所有 /root/workspace/corpus 路径

步骤四：运行训练与分阶段执行

一键训练：

```
bash run.sh
```

分阶段执行：通过修改 stage_run.sh 的 start_stage 与 end_stage 控制阶段（数据合成 =1，数据准备 =2，训练 =3，测试与报告 =4）。以下给出四种示例（节选，与原始等价）：

```
#!/bin/bash
. ./path.sh
source ./path.sh

TOP_CONFIG="./top_conf.yaml"
PROJECT_CONFIG="./conf/prj_conf.yaml"
CONFIG_DIR="./conf/train_conf"

# 只执行数据合成阶段
version='v0.1.1'
start_stage=1
end_stage=1

python -u auto_finetune.py \
    --top_config "$TOP_CONFIG" \
    --project_config "$PROJECT_CONFIG" \
    --strategy_config "$CONFIG_DIR/${version}.yaml" \
```

(续下页)

(接上页)

```
  --start_stage "$start_stage" \
  --end_stage "$end_stage"
```

（其余三段对应阶段 2、3、4 的脚本同上，仅变更 start_stage/end_stage 值，内容与原文保持一致。）

运行结果：

- 训练完成后生成模型与测试报告，Excel 报告需人工查看

3.4 模型文件说明

模型文件类型：

- 选择一：model_b → 支持 GX8002b，输出 model_grus.h;
- 选择二：model_g → 支持 GX8002D，输出 model_grus.h; 支持 GX8008C 编译，输出 olab_nn.c,weight.c 等;
- 选择三：model_f → 支持 GX8003；支持 GX8005/GX8006（输出 model_fornax.h 与 mean_std.txt）支持 GX8008C 编译, 输出 olab_nn.c,weight.c 等;
- 选择四：model_k → 支持 GX8302B（输出 model_grus.h、model_apus.h 与 mean_std.txt）；
- 选择五：model_t（仅大小模型）→ 支持 GX8002b，输出 model.h;

模型输出目录：output/pub_model

测试报告路径示例：

- /home/pub/workspace/kws/dryer_prj/version/v0.1.1/result/ori/*excel/default/*.xlsx
- /home/pub/workspace/kws/dryer_prj/version/v0.2.1/result/ori/*excel/default/*.xlsx
- /home/pub/workspace/kws/dryer_prj/version/v0.3.1/result/ori/*excel/default/*.xlsx
- /home/pub/workspace/kws/dryer_prj/version/v0.4.1/result/ori/*excel/default/*.xlsx

注意：系统会根据测试报告选择最优模型保存到 output/pub_model，仍建议人工核验。

3.5 自有数据训练配置

Alchemy V3.6 支持使用客户自有数据进行模型训练。客户可准备自有音频数据，通过提供的脚本生成支持的数据库格式，并完成训练。

3.5.1 数据准备要求

1) 数据质量要求
- 训练数据与测试数据严格分离
- 测试数据不得包含训练数据对应说话人
- 确保测试结果具有代表性

2) 音频格式要求
- 采样率：16kHz
- 编码：16-bit signed int
- 通道：单声道（1）
- 时长：≤ 7.6 秒
- 格式：WAV

3) 音频格式转换示例

```
sox input.wav -c 1 -b 16 -r 16000 output.wav
```

4）数据组织结构

训练数据目录结构：

```
train_wav/
├── 指令词1/
│   ├── file1.wav
│   ├── file2.wav
│   └── ...
├── 指令词2/
│   ├── file1.wav
│   ├── file2.wav
│   └── ...
└── ...
```

测试数据目录结构：

```
test_wav/
├── 指令词1/
│   ├── file1.wav
│   ├── file2.wav
│   └── ...
├── 指令词2/
│   ├── file1.wav
│   ├── file2.wav
│   └── ...
└── ...
```

3.5.2 数据列表生成

使用 gen_list_with_dir.sh 脚本生成列表：

- 脚本位置：~/workspace/corpus/gen_list_with_dir.sh
- 基本用法：

```
./gen_list_with_dir.sh [输入目录] [输出目录]
```

示例：

```
cd ~/workspace/corpus
./gen_list_with_dir.sh ./train_wav ~/workspace/corpus/common_data/hifi_valle_database/hifi/train/list/asr
./gen_list_with_dir.sh ./test_wav  ~/workspace/corpus/common_data/hifi_valle_database/hifi/test/list/asr
```

脚本功能说明：

- 扫描一级子文件夹；为每个子文件夹生成 wav 列表
- 仅处理 wav 文件；跳过空文件夹或包含子目录的文件夹
- 列表内容格式：绝对路径, 标签名

3.5.3 配置文件修改

本小节将所有需要修改的字段按“文件”集中说明，避免分散。

- 修改 top_conf.yaml（项目基本配置）

```
cd ~/workspace/alchemy/example_0
vi top_conf.yaml
```

示例修改：

```
cmd_conf:
  main_cmd:
  - 您的指令词1
  - 您的指令词2|指令词3
  other_cmd:
  - 其他指令词1|指令词2
  - 其他指令词3
## 指令词仅支持中文，不包含任何符号或阿拉伯数字
model_conf:
  chip: gx8002
  model_name: model_f
project_conf:
  gpu_id: 0
  project_name: my_project
```

- 修改 conf/prj_conf.yaml（数据路径与合并量配置）

```
vi conf/prj_conf.yaml
```

关键字段：

```
database_conf:
  train_database_conf:
    hifi_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/hifi/train/list/asr'
    valle_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/valle/train/list/asr'
    cosyvoice_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/cosyvoice/train/list/asr
↪'
  test_database_conf:
    hifi_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/hifi/test/list/asr'
    valle_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/valle/test/list/asr'
    cosyvoice_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/cosyvoice/test/list/asr'
↪
```

自有数据合并量（merge_num/merge_max_wav_num）：

```
train_conf:
  merge_conf:
    # 设置为“自有数据的数据量”（条数）。示例：自有数据 1000 条
    merge_max_wav_num: 1000
```

- 修改 conf/train_conf/v0.*.yaml（训练增强/域增强/部署参数/target_num）

```
vi conf/train_conf/v0.1.1.yaml
```

v0.1.1.yaml 需要集中修改与检查：

```yaml
dataset_conf:
  train_aug_conf:
    noise_dir: '/root/workspace/corpus/common_data/noise/add_noise'
    ground_noise_dir: '/root/workspace/corpus/common_data/noise/ground_noise'
    reverb_dir: '/root/workspace/corpus/common_data/RIR'
    white_noise_dir: '/root/workspace/corpus/common_data/noise/white_noise'
    # 将 target_num 修改为 2 × 自有数据量（示例：自有 1200 条 → 2400）
    target_num: 2400
  domain_aug_conf:
    noise_dir: '/root/workspace/corpus/common_data/noise/add_noise'
    ground_noise_dir: '/root/workspace/corpus/common_data/noise/domain_noise'
    reverb_dir: '/root/workspace/corpus/common_data/RIR'
    white_noise_dir: '/root/workspace/corpus/common_data/noise/white_noise'
    # 同样设置为 2 × 自有数据量
    target_num: 2400
  common_tfr_conf:
    asr_tfr_dir: ['/root/workspace/corpus/common_data/ASR']
deploy_conf:
  neg_path: '/root/workspace/corpus/common_data/test_set/neg_8002'
```

在 conf/train_conf/v0.2.1.yaml 执行与上面一致的修改（两处 target_num 皆为 2 × 自有数据量），并同步检查噪声/ASR 路径与 deploy_conf.neg_path。

统一提醒：

- 将所有 /root 替换为实际用户路径（如 /home/username 或 $HOME）
- 使用自有数据时，路径需指向生成的 .list 文件
- 其他配置参数一般无需修改

注：上文已集中给出 top_conf.yaml、conf/prj_conf.yaml、v0.*.yaml 的所有修改点。后续章节仅作引用，不再重复细节。

3.5.4 数据量配置说明

仅使用自有数据训练时（减少克隆数据）：

```yaml
data_prepare_conf:
  train_conf:
    hifi_conf:
      hifi_max_wav_num: 1000
    cosyvoice_conf:
      cosyvoice_max_wav_num: 10
    valle_conf:
      valle_max_wav_num: 10
  test_conf:
    hifi_conf:
      hifi_max_wav_num: 100
    cosyvoice_conf:
      cosyvoice_max_wav_num: 10
```

(续下页)

(接上页)

```
    valle_conf:
      valle_max_wav_num: 10
```

大量自有语料训练（全部纳入）：

```
dataset_conf:
  train_aug_conf:
    hifi_max_wav_num: 1200
```

注意：hifi_max_wav_num 应设置为自有指令词数据的最大 wav 数量；数据量特别大时建议分批训练。

额外必改参数（目标条目数 target_num）：

- 文件：alchemy/example_0/conf/train_conf/v0.1.1.yaml
- 文件：alchemy/example_0/conf/train_conf/v0.2.1.yaml
- 位置 1：dataset_conf.train_aug_conf.target_num
- 位置 2：dataset_conf.domain_aug_conf.target_num

设置规则（两处都要改）：

- train_aug_conf.target_num = 2 × 自有数据量
- domain_aug_conf.target_num = 自有数据量

示例（假设自有数据量为 1200 条）：

```
dataset_conf:
  train_aug_conf:
    # ... 其他字段
    target_num: 2400
  domain_aug_conf:
    # ... 其他字段
    target_num: 1200
```

同时在 conf/prj_conf.yaml 中设置：

```
data_prepare_conf:
  train_conf:
    hifi_conf:
      hifi_max_wav_num: 1200  # 等于自有数据量
    cosyvoice_conf:
      cosyvoice_max_wav_num: 10
    valle_conf:
      valle_max_wav_num: 10
    merge_conf:
      merge_max_wav_num: 1200   # 等于自有数据量
  test_conf:
    hifi_conf:
      hifi_max_wav_num: 100
    cosyvoice_conf:
      cosyvoice_max_wav_num: 10
    valle_conf:
      valle_max_wav_num: 10
    merge_conf:
      merge_max_wav_num: 100   # 等于自有数据量
```

重要提醒：

- hifi_max_wav_num 设置为自有指令词数据的最大 wav 数量
- 数据量特别大时建议分批训练

3.6 GX8008C 双通道测试功能

当在 top_conf.yaml 的 model_conf.chip 设置为 gx8008c 时，系统支持双通道测试。根据解码需求选择以下任一配置方式：

1) 多通道 ICA + CTC + GXDecoder 解码（在 top_conf.yaml 追加）

```yaml
# top_conf.yaml
model_conf:
  chip: gx8008c  ## 若chip为gx8008c，则默认合成双通道测试集
  model_name: model_f

deploy_conf:
  neg_path: '/root/workspace/corpus/common_data/test_set/neg_8008'  ## 双通道误唤醒测试集
  pos_path: '${project_corpus_path}/test/GAT'
  try_name: 'default'
  gxdecoder_flag: true   ## 使用 gxdecoder解码
  test_set_flag: true
  average_ckpt_flag: true
  ICA: true      ## 使用ICA 前处理
  multi_channel_flag: true ## 使用多通道合并解码

report_conf:
  neg_act_num: {"ori_kws_name":3,"other_words":8,}
  merge_FNR_words_list: [['',''],]
  ICA: true    ## 使用ICA 前处理
  multi_channel_flag: true ## 使用多通道合并解码
```

2) 仅 ICA + CTC 解码（在 top_conf.yaml 追加）

```yaml
deploy_conf:
  neg_path: '/root/workspace/corpus/common_data/test_set/neg_8008'  ## 双通道误唤醒测试集
  pos_path: '${project_corpus_path}/test/GAT'
  try_name: 'default'
  gxdecoder_flag: false   ## 不使用gxdecoder解码，而仅用ctc解码
  test_set_flag: true
  average_ckpt_flag: true
  ICA: true  ## 使用ICA 前处理
  multi_channel_flag: true ## 使用多通道合并解码

report_conf:
  ICA: true  ## 使用ICA 前处理
  multi_channel_flag: true ## 使用多通道合并解码
```

3) 替代方案：在 conf/train_conf/v0.*.yaml 修改对应版本

```yaml
# conf/train_conf/v0.1.1.yaml
deploy_conf:
```

(续下页)

(接上页)

```
  neg_path: '/root/workspace/corpus/common_data/test_set/neg_8008'
  pos_path: '${project_corpus_path}/test/GAT'
  try_name: 'default'
  gxdecoder_flag: true
  test_set_flag: true
  average_ckpt_flag: true
  ICA: true
  multi_channel_flag: true ## 使用多通道合并解码
report_conf:
  ICA: true
  multi_channel_flag: true ## 使用多通道合并解码
```

必改与注意：

- 将 neg_path 由 neg_8002 改为 neg_8008，才可进行负样本的双通道推理解码
- 根据需求设置 gxdecoder_flag 与 ICA,multi_channel_flag，控制 gxdecoder 与 ICA 的开关。
- 一个 YAML 文件对应一个输出模型

3.7 完整流程与注意事项

本节提供从数据准备到训练产出的完整、可执行流程，含四个阶段脚本的完整示例，以及关键注意点与验收项。

3.7.1 准备数据与目录

1）创建目录并放置音频（参考 3.5.1 的格式要求）

```
mkdir -p ~/my_audio_data/train_wav ~/my_audio_data/test_wav
# 将音频按指令词分类放入 train_wav/ 与 test_wav/ 的各子目录
```

2）若需格式统一（16kHz、单声道、16-bit），可批量转换

```
for file in ~/my_audio_data/train_wav/*/*.wav; do
  sox "$file" -c 1 -b 16 -r 16000 "${file%.wav}_converted.wav"
  mv "${file%.wav}_converted.wav" "$file"
done
for file in ~/my_audio_data/test_wav/*/*.wav; do
  sox "$file" -c 1 -b 16 -r 16000 "${file%.wav}_converted.wav"
  mv "${file%.wav}_converted.wav" "$file"
done
```

3.7.2 生成数据列表（.list）

使用 gen_list_with_dir.sh 生成训练/测试列表：

```
cd ~/workspace/corpus
./gen_list_with_dir.sh ~/my_audio_data/train_wav ~/workspace/corpus/common_data/hifi_valle_database/hifi/
↪train/list/asr
./gen_list_with_dir.sh ~/my_audio_data/test_wav  ~/workspace/corpus/common_data/hifi_valle_database/hifi/test/
↪list/asr
```

验证输出：每个指令词一个 .list 文件，内容为 绝对路径, 标签名。

3.7.3 修改项目配置

1）top_conf.yaml（项目名、芯片、模型、指令词）

```
cd ~/workspace/alchemy/example_0
vi top_conf.yaml
```

示例字段：

```
model_conf:
  chip: gx8002
  model_name: model_f
project_conf:
  project_name: my_project
  gpu_id: 0
```

2）conf/prj_conf.yaml（训练/测试列表路径）

```
vi conf/prj_conf.yaml
```

关键字段：

```
database_conf:
  train_database_conf:
    hifi_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/hifi/train/list/asr'
  test_database_conf:
    hifi_asr_list_lib: '/root/workspace/corpus/common_data/hifi_valle_database/hifi/test/list/asr'
```

请将所有 /root 替换为实际用户路径（如 /home/username 或 $HOME）。

自有数据合并配置（按自有数据量设置）：

```
train_conf:
  merge_conf:
    # 设置为“自有数据的数据量”（条数）
    merge_max_wav_num: 1200
```

3）conf/train_conf/v0.*.yaml（数据增强、噪声目录、ASR TFR、部署参数、target_num）

```
vi conf/train_conf/v0.1.1.yaml
```

示例检查点：

```
dataset_conf:
  train_aug_conf:
    noise_dir: '/root/workspace/corpus/common_data/noise/add_noise'
    ground_noise_dir: '/root/workspace/corpus/common_data/noise/ground_noise'
    reverb_dir: '/root/workspace/corpus/common_data/RIR'
    white_noise_dir: '/root/workspace/corpus/common_data/noise/white_noise'
    # 将 target_num 修改为 2 × 自有数据量
    target_num: 2400
  common_tfr_conf:
    asr_tfr_dir: ['/root/workspace/corpus/common_data/ASR']
deploy_conf:
  neg_path: '/root/workspace/corpus/common_data/test_set/neg_8002'
```

同样在 domain_aug_conf 中将 target_num 设置为 2 × 自有数据量；并在 v0.2.1.yaml 执行相同修改。

3.7.4 启动训练（全流程）

```
cd ~/workspace/alchemy/example_0
bash run.sh
```

产出位置：

- 模型：output/pub_model
- 日志：output/log
- 报告：version/vX.Y.Z/result/ori/*excel/default/*.xlsx

3.7.5 分阶段执行脚本（四个完整示例）

脚本位置：stage_run.sh。通过 start_stage 与 end_stage 控制阶段（1 数据合成、2 数据准备、3 模型训练、4 测试与报告）。以下四段为可直接复制的完整示例：

1）仅执行数据合成（stage 1-1）

```
#!/bin/bash
. ./path.sh
source ./path.sh

TOP_CONFIG="./top_conf.yaml"
PROJECT_CONFIG="./conf/prj_conf.yaml"
CONFIG_DIR="./conf/train_conf"

version='v0.1.1'  ## 也可为其他三个版本： v0.2.1,v0.3.1,v0.4.1
start_stage=1
end_stage=1

python -u auto_finetune.py \
  --top_config "$TOP_CONFIG" \
  --project_config "$PROJECT_CONFIG" \
  --strategy_config "$CONFIG_DIR/${version}.yaml" \
  --start_stage "$start_stage" \
  --end_stage "$end_stage"
```

2）仅执行数据准备（stage 2-2）

```
#!/bin/bash
. ./path.sh
source ./path.sh

TOP_CONFIG="./top_conf.yaml"
PROJECT_CONFIG="./conf/prj_conf.yaml"
CONFIG_DIR="./conf/train_conf"

version='v0.1.1' ## 也可为v0.2.1
start_stage=2
end_stage=2

python -u auto_finetune.py \
```

(续下页)

(接上页)

```
  --top_config "$TOP_CONFIG" \
  --project_config "$PROJECT_CONFIG" \
  --strategy_config "$CONFIG_DIR/${version}.yaml" \
  --start_stage "$start_stage" \
  --end_stage "$end_stage"
```

3）仅执行模型训练（stage 3-3）

```
#!/bin/bash
. ./path.sh
source ./path.sh

TOP_CONFIG="./top_conf.yaml"
PROJECT_CONFIG="./conf/prj_conf.yaml"
CONFIG_DIR="./conf/train_conf"

version='v0.1.1'  ## 也可为其他三个版本： v0.2.1,v0.3.1,v0.4.1
start_stage=3
end_stage=3

python -u auto_finetune.py \
  --top_config "$TOP_CONFIG" \
  --project_config "$PROJECT_CONFIG" \
  --strategy_config "$CONFIG_DIR/${version}.yaml" \
  --start_stage "$start_stage" \
  --end_stage "$end_stage"
```

4）仅执行测试与报告（stage 4-4）

```
#!/bin/bash
. ./path.sh
source ./path.sh

TOP_CONFIG="./top_conf.yaml"
PROJECT_CONFIG="./conf/prj_conf.yaml"
CONFIG_DIR="./conf/train_conf"

version='v0.1.1'  ## 也可为其他三个版本： v0.2.1,v0.3.1,v0.4.1
start_stage=4
end_stage=4

python -u auto_finetune.py \
  --top_config "$TOP_CONFIG" \
  --project_config "$PROJECT_CONFIG" \
  --strategy_config "$CONFIG_DIR/${version}.yaml" \
  --start_stage "$start_stage" \
  --end_stage "$end_stage"
```

执行前检查清单（强烈建议逐项核对）：

- target_num：在 v0.1.1.yaml 与 v0.2.1.yaml 的 dataset_conf.train_aug_conf.target_num 均为 2 × 自有数据量，dataset_conf.domain_aug_conf.target_num 均为 1 × 自有数据量
- 自有数据列表路径：conf/prj_conf.yaml 的 train/test 列表目录均指向实际生成的 .list

- 绝对路径：所有 conf/* 中不再残留 /root/... 示例路径，均替换为实际用户路径
- neg_path：deploy_conf.neg_path 芯片匹配（如 GX8008C 使用 neg_8008）
- 环境变量：source ~/.bashrc 后 conda 可用，四个环境完整解压
- 权限：chmod +x 赋权后脚本可执行

3.7.6 验收与常见排错入口

验收：

- output/pub_model 出现对应芯片/模型的发布文件，在 pub.yaml 文件中即为当前发布文件的相关信息。
- version/vX.Y.Z/result/.../*.xlsx 报告生成且指标合理

如遇问题：先按第 4 章 Q&A 的顺序排查（权限、conda、环境解压、缓存、路径）。

4 Alchemy 系统常见问题

4.1 常见问题（含详细处理步骤）

1. 权限问题
   - 现象：运行脚本时提示权限不足
   - 处理：

```
chmod +x alchemy/example_*/run.sh
```

2. conda 命令不可用或环境变量失效
   - 现象：执行 conda 提示 command not found
   - 处理：

```
source ~/.bashrc
```

3. Python 环境缺失
   - 现象：激活 conda 环境失败，提示环境不存在
   - 排查：确认以下环境均已解压到 ~/miniforge3/envs/
     - py36_tf115
     - th_12
     - SenseVoice_py311
     - cosyvoice
4. 路径配置问题
   - 现象：训练时提示找不到文件或路径错误
   - 排查：检查 alchemy/example_*/conf/*.yaml 中的绝对路径是否已替换为实际用户路径
5. 环境安装失败
   - 现象：自动安装脚本执行失败，依赖库安装不完整
   - 处理：

```
source ~/.bashrc
conda activate py36_tf115
cd ~/workspace/alchemy/alchemy/tools/logfbank
python setup.py develop
pip install npu-compiler==1.6.5
```

6. 数据准备失败
   - 问题 1：模型缓存问题（典型日志包含下载 encodec_24khz .th 失败）
     - 处理：

```
rm -r ~/.cache
mkdir ~/.cache
cp -r ~/workspace/corpus/cache/* ~/.cache/
```

- 问题 2：环境解压不完整
  - 处理：

```
rm -r ~/miniforge3/envs/th_12  ~/miniforge3/envs/SenseVoice_py311
cd ~/downloads
tar -zxvf th_12.tar.gz -C  ~/miniforge3/envs/
tar -zxvf SenseVoicepy311.tar.gz -C  ~/miniforge3/envs/
```

7. 测试集生成问题
   - 现象：断言 len(kws_num.flatten(...)) == len(excel_params['wav']['pos']) 失败
   - 原因：同一指令词同一说话人的测试集生成了两个
   - 处理：删除 corpus/test/GAT，重新生成测试集
   - 可用脚本（在 stage_run.sh 指定阶段 2）：

```
# 重新生成测试集
version='v0.3.1'
start_stage=2
end_stage=2
```

8. 误唤醒个数修改
   - 方法一：在 top_conf.yaml 追加 report_conf.neg_act_num
   - 方法二：在 conf/train_conf/v0.*.yaml 的 report_conf 中配置
   - 示例：

```
report_conf:
  neg_act_num: {"ori_kws_name":3,"other_words":8,"你好小树":1,}
```

9. 阈值修改
   - 方法一（推荐）：在指定版本 conf/train_conf/v0.*.yaml 中配置
   - 方法二：在 top_conf.yaml 中追加全局 report_conf.want_THR
   - 示例：

```
report_conf:
  want_THR: {"ori_kws_name":84.8,"你好小树":85.5}
```

4.2 技术支持

如遇问题：

1) 查看日志输出
2) 检查环境变量与路径
3) 升级客户参考升级指南
4) 联系支持，并提供错误信息与环境状态

重要提醒：

- 新客户按完整流程安装
- V3.0+ 升级客户可用简化流程，升级前做好备份
- 先用 example_local 验证环境，再用 example_0 做正式项目

杭州国芯微电子股份有限公司

中国 杭州总部：
电话（TEL）：86+571-88156088
传真（FAX）：86+571-88156083
地址（ADD）：中国浙江省杭州市文三路 90 号东部软件园科技大厦五层 邮编：310012

中国 深圳分部：
电话（TEL）：86+755-86562061
地址（ADD）：中国广东省深圳市南山区粤海街道高新南一道 008 号创维大厦 A 座 801 室