# 代码分析报告

# 一、摘要

## 代码结构概览

本次分析涵盖了chordPrediction仓库master分支下的所有模块。主模块为VirtualKeyboard.py，该模块定义了一个支持MIDI输入和键盘映射的系统，用于和弦序列预测，并支持情绪和风格标签作为上下文。其他文件包括ChordHMM.py、MatrixView.py、utils.py等，这些模块提供了与Viterbi算法、模型训练、图形界面相关的核心功能。这些模块之间的关系图如下：

VirtualKeyboard.py - 统筹管理用户界面及主要功能  
ChordHMM.py - 实现HMM模型的训练与预测  
MatrixView.py - 提供图形界面中的图像显示功能  
utils.py - 提供辅助功能，如文件操作、日志记录

## 核心模块和函数

### VirtualKeyboard.py

#### 核心函数：

* initUI: 用于初始化用户界面，设置窗口大小和标题，根据是否使用MIDI键盘显示不同的标题，并创建各种控件和布局。
* shaderLists: 用于在GUI中创建一个包含标注库和历史记录的列表视图，并将对应的文本内容显示在堆叠窗口中。
* updateChords: 用于更新和弦序列，并根据当前和弦预测下一个和弦。
* pressingEvent: 用于检测当前按下的琴键，并根据按下的琴键数量和音符组合来确定当前的和弦。
* reShaderLabels: 用于重新设置标签的布局，清空现有的布局并重新添加多选框。

### ChordHMM.py

#### 核心函数：

* baum\_welch: 实现Baum-Welch算法，用于训练隐马尔可夫模型（HMM）的参数。

### MatrixView.py

#### 核心函数：

* update\_image: 根据当前和预测的和弦序列更新显示的图像，通过绘制和弦转换矩阵的热图来实现。

## 代码设计风格分析

1. **命名规范**：采用了小写字母加下划线（snake\_case）的命名风格，如initUI、shaderLists等，符合PEP 8 Python的命名规范。
2. **一致性**：模块间调用、变量命名、函数参数都较为一致，如频繁使用self作为实例变量，baum\_welch函数的max\_iters参数等。
3. **封装与抽象程度**：类和函数设计较为合理，各模块内封装了具体业务逻辑，如VirtualKeyboard.py中定义了用户界面的主要功能。
4. **模块职责划分**：模块职责分工明确，每个模块专注于一个功能，如VirtualKeyboard.py负责用户界面，ChordHMM.py负责模型训练。

## 潜在问题

1. **资源释放不当**：在shaderLists和reShaderLabels中，scroll\_content的子控件被释放后，可能不会立即释放相关的内存资源，需要考虑更好的内存管理。
2. **异常未处理**：例如，shadersLists中的filePath函数调用可能会因为路径不存在而导致异常，需要在使用前添加异常处理逻辑。
3. **重复或冗余的逻辑**：函数间存在重复的代码逻辑，如两次调用了reShaderLists，可以考虑将重复的代码提取到一个独立的函数中。
4. **低效的实现**：onLabel和onRecord函数中，每次操作都刷新列表和标签，即使内容未变化，可能导致不必要的性能开销。

## 重构建议

1. **减少重复代码**：将重复的代码提取到一个独立的函数中，如将列表和标签刷新的逻辑提取到一个函数中。
2. **内存管理**：在释放控件时，确保相关的内存资源也被释放，可以使用Python的gc模块进行垃圾回收。
3. **异常处理**：在调用可能抛出异常的函数时，添加异常处理逻辑，如在filePath中添加except语句。
4. **使用图表工具**：对于复杂的算法如Baum-Welch，可以考虑使用图表工具（如Matplotlib）来可视化迭代过程和结果，提高调试和理解的效率。

## 测试情况

* **测试覆盖范围**：当前的测试主要覆盖了用户界面和财务系统的功能。
* **测试健壮性**：虽然测试用例覆盖了正常情况，但缺乏对异常情况和边界情况的测试，建议增加单元测试，确保代码的健壮性。
* **验证功能**：测试主要通过验证界面元素和内部状态的变化来判断功能是否正常，建议加入更多的日志记录和断言，以验证内部逻辑的正确性。
* **边界情况和异常情况**：部分关键函数没有覆盖边界情况和异常情况的测试用例，建议加入更多的测试用例，确保代码在所有边界情况和异常情况下都能正常运行。

通过以上分析，可以看出chordPrediction仓库的代码整体结构清晰，设计合理，但由于一些潜在问题和冗余逻辑，需要进行相应的重构和优化，以提高代码质量和性能。

# 二、附录明细

## 基础信息

* 仓库名称：chordPrediction
* 仓库描述：和弦预测
* 仓库分支：master
* 仓库地址：https://github.com/kinglegendzzh/chordPrediction
* 项目根路径：/Users/apple/Public/generates-git/chordPrediction
* 分析的目标子路径：.

## 函数信息

### initUI (VirtualKeyboard.py)

* 行号位置：99-248
* 重要性评分：137.00

**代码片段：**

def initUI(self):  
 # 设置窗口大小和标题  
 self.setFixedSize(1200, 850)  
 self.setWindowTitle('智能化音乐创作工具')  
 if self.NoneMIDI:  
 logging.info('使用<英文键盘>映射模式')  
 title = QLabel("使用<英文键盘>映射模式")  
 else:  
 logging.info('成功接入MIDI键盘')  
 title = QLabel('虚拟MIDI键盘')  
 title.setAlignment(Qt.AlignCenter)  
  
 grid = QGridLayout()  
 grid.setObjectName("midi")  
 grid.setHorizontalSpacing(0)  
 grid.setVerticalSpacing(0)  
 grid.setAlignment(Qt.AlignTop)  
 for i in range(len(self.keys)):  
 key = QPushButton('')  
 key.setObjectName(self.values[i].name)  
 if self.keys[i] == 'w':  
 key.setFixedSize(self.key\_width, self.white\_key\_height)  
 self.changeWhiteSheet(key)  
 else:  
 key.setFixedSize(int(self.key\_width / 2), self.black\_key\_height)  
 self.changeBlackSheet(key)  
 # key.clicked.connect(lambda state, i=i: self.onPressed(i))  
 grid.addWidget(key, 0, i)  
 self.keys\_button.append(key)  
  
 grid.setAlignment(Qt.AlignCenter)  
  
 ch = QLabel('识别当前和弦: ')  
 ch.setAlignment(Qt.AlignCenter)  
 ch.setObjectName("chords")  
 self.vbox.addWidget(ch)  
  
 ch = QLabel('预测下一个和弦: ')  
 ch.setAlignment(Qt.AlignCenter)  
 ch.setObjectName("next")  
 self.vbox.addWidget(ch)  
  
 self.buttonV = QPushButton("打开概率转移矩阵分析图", self)  
 self.buttonV.clicked.connect(self.open\_image\_window)  
 self.vbox.addWidget(self.buttonV)  
  
 # 初始化和弦序列  
 chTitle = QLabel("实时记录最新十条和弦(松开琴键以写入,点击和弦以移出序列)：")  
 chTitle.setObjectName("chordsQueueTitle")  
 self.vbox.addWidget(chTitle)  
 self.chordsQueue = QHBoxLayout()  
 self.chordsQueue.setSpacing(0)  
 self.chordsQueue.setObjectName("chordsQueue")  
 # 初始化渲染和弦序列  
 for i in range(0, self.MAX\_QUEUE):  
 ql = QPushButton("")  
 ql.setObjectName("ch" + str(i))  
 font = QFont('Calibri', 16)  
 ql.setFont(font)  
 # self.reShaderButton(ql)  
 ql.clicked.connect(lambda state, i=i: self.onPressed(i))  
 self.chordsQueue.addWidget(ql)  
 self.vbox.addLayout(self.chordsQueue)  
  
 # 创建3个单选按钮  
 self.radio\_btn1 = QRadioButton('一阶')  
 self.radio\_btn2 = QRadioButton('二阶')  
 self.radio\_btn3 = QRadioButton('三阶')  
 # 设置默认选中  
 self.radio\_btn2.setChecked(True)  
 self.hbox\_radio = QHBoxLayout()  
 self.hbox\_radio.setObjectName("radio")  
 self.hbox\_radio.addWidget(QLabel('预测模型准确度'))  
 self.hbox\_radio.addWidget(self.radio\_btn1)  
 self.hbox\_radio.addWidget(self.radio\_btn2)  
 self.hbox\_radio.addWidget(self.radio\_btn3)  
 self.vbox.addLayout(self.hbox\_radio)  
  
 ch = QLabel('预选音乐风格/标签（上下滑动选择更多）: ')  
 ch.setAlignment(Qt.AlignCenter)  
 self.vbox.addWidget(ch)  
 # 遍历目录下的文件，添加到多选框中  
 self.labelBox = QVBoxLayout()  
 self.labelBox.setObjectName("labelO")  
 self.scroll\_area = QScrollArea()  
 self.scroll\_area.setWidgetResizable(True)  
 self.scroll\_area.setObjectName("scroll")  
 self.scroll\_content = QWidget(self.scroll\_area)  
 self.scroll\_content.setObjectName("labelContent")  
 self.scroll\_content.setLayout(self.labelBox)  
 self.scroll\_area.setWidget(self.scroll\_content)  
 labelhbox = QHBoxLayout()  
 count = 0 # 记录已添加的多选框数量  
 for file\_name in os.listdir(filePath('labels/')):  
 name, extension = os.path.splitext(file\_name)  
 check\_box = QCheckBox(name, self)  
 check\_box.setText(name)  
 labelhbox.addWidget(check\_box)  
 count += 1  
 if count >= 5: # 超过x个数量，则另起一行继续水平排列  
 self.labelBox.addLayout(labelhbox)  
 labelhbox = QHBoxLayout()  
 count = 0  
 if count > 0: # 处理剩余不到x个的多选框  
 self.labelBox.addLayout(labelhbox)  
 self.vbox.addWidget(self.scroll\_area)  
  
 hbox2 = QHBoxLayout()  
 hbox3 = QHBoxLayout()  
 hbox4 = QHBoxLayout()  
 hbox5 = QHBoxLayout()  
 self.label\_1 = QLabel("序列命名：", self)  
 self.edit\_1 = QLineEdit(self)  
 self.label\_2 = QLabel("标签：", self)  
 self.edit\_2 = QLineEdit(self)  
 self.label\_3 = QLabel("备注：", self)  
 self.edit\_3 = QLineEdit(self)  
 # 创建“保存”和“标注”按钮  
 self.save\_btn = QPushButton("保存序列", self)  
 ...

* 功能描述： 该函数用于初始化用户界面，设置窗口大小和标题，根据是否使用MIDI键盘显示不同的标题，并创建各种控件和布局，包括按键、标签、单选按钮、多选框、文本框和按钮等。这些控件用于处理音乐创作、和弦识别、预测和保存等功能。
* 实现流程： 设置窗口大小和标题。 根据是否使用MIDI键盘显示不同的标题。 创建按键和布局，包括白键和黑键的按钮。 添加识别和预测和弦的标签。 添加打开概率转移矩阵分析图的按钮。 初始化和弦序列的显示和操作。 创建预测模型准确度的单选按钮。 添加预选音乐风格/标签的多选框。 添加序列命名、标签和备注的文本框和按钮。 添加所有库的列表视图和堆叠窗口。 设置状态展示框，显示当前监听状态。 将所有控件和布局添加到主布局中，并设置主布局。 初始化和弦序列的渲染和操作。 处理按键事件，记录按键信息并更新队列。 处理记录操作，将队列数组和编辑框文本写入文件。 处理标签文件的创建和更新，将QUEUE数组内容写入.model文件。 清空队列并重置所有按钮的文本。 在GUI中创建标注库和历史记录的列表视图，并实现列表和堆叠窗口之间的交互。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： setFixedSize,setWindowTitle,info,QLabel,setAlignment,QGridLayout,setObjectName,setHorizontalSpacing,setVerticalSpacing,QPushButton,changeWhiteSheet,changeBlackSheet,addWidget,connect,QHBoxLayout,setSpacing,QFont,setFont,onPressed,addLayout,QRadioButton,setChecked,QVBoxLayout,QScrollArea,setWidgetResizable,QWidget,setLayout,setWidget,listdir,filePath,splitext,QCheckBox,setText,QLineEdit,onRecord,onLabel,onClear,shaderLists,setStyleSheet,
* 内部依赖描述：
  + changeWhiteSheet: 根据传入的isGray参数，改变按钮的样式。如果isGray为True，则设置按钮的边框为灰色，圆角为3px，并且背景颜色为#878787。如果isGray为False，则设置按钮的边框为灰色，圆角为3px，并且背景颜色为self.white\_color。
  + changeBlackSheet: 根据传入的isGray参数，改变按钮的样式。如果isGray为True，则按钮的背景颜色为#787878；如果isGray为False，则按钮的背景颜色为self.black\_color。
  + onPressed: 该函数用于处理按键事件，当按键被按下时，记录按键信息，并从队列中移除指定索引位置的元素。
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。
  + onRecord: 该函数用于处理记录操作，当编辑框1不为空时，根据编辑框1的文本创建或追加文件，并将队列数组和编辑框3的文本写入文件。如果文件已存在，则追加内容；否则，创建新文件并写入初始内容。最后，刷新列表以显示新的着色器列表。
  + onLabel: 该函数用于处理标签文件的创建和更新。当输入框edit\_2不为空时，它会根据输入的文件名列表创建或更新相应的.model文件，并将QUEUE数组的内容以特定格式写入文件。如果文件已存在，则追加内容；如果不存在，则创建新文件并写入初始内容。最后，刷新着色器列表和标签。
  + onClear: 该函数用于清空队列并重置所有按钮的文本。
  + shaderLists: 该函数用于在GUI中创建一个包含标注库和历史记录的列表视图，并将对应的文本内容显示在堆叠窗口中。它会读取指定目录下的文件，并将文件名和内容分别添加到列表和堆叠窗口中，同时实现列表和堆叠窗口之间的交互。

### shaderLists (VirtualKeyboard.py)

* 行号位置：255-309
* 重要性评分：35.50

**代码片段：**

def shaderLists(self, vbox, listWidget=None, stackedWidget=None):  
 type = 0  
 # 读取模型  
 if listWidget is None and stackedWidget is None:  
 listWidget = QListWidget(self)  
 stackedWidget = QStackedLayout()  
 listWidget.setObjectName("listO")  
 stackedWidget.setObjectName("stackO")  
 listWidget.setMaximumWidth(250)  
 type = 1  
 directory\_labels = "labels/"  
 directory\_records = "records/"  
 for file\_name in os.listdir(filePath(directory\_labels)):  
 item = '标注库- ' + str(file\_name)  
 listWidget.addItem(item)  
 text = ""  
 # 创建一个多行文本域控件  
 text\_edit = QTextEdit(text, self)  
 # 设置文本域为只读  
 text\_edit.setReadOnly(True)  
 cursor = text\_edit.textCursor()  
 with open(filePath(directory\_labels) + str(file\_name), 'r', encoding='utf-8') as f:  
 lineNum = 1  
 for line in f:  
 if lineNum != 1:  
 text += line + '\n'  
 cursor.insertText(line)  
 cursor.insertText('\n')  
 lineNum += 1  
  
 stackedWidget.addWidget(text\_edit)  
 for file\_name in os.listdir(filePath(directory\_records)):  
 item = '历史记录- ' + str(file\_name)  
 listWidget.addItem(item)  
 text = ""  
 # 创建一个多行文本域控件  
 text\_edit = QTextEdit(text, self)  
 # 设置文本域为只读  
 text\_edit.setReadOnly(True)  
 cursor = text\_edit.textCursor()  
 with open(filePath(directory\_records) + str(file\_name), 'r', encoding='utf-8') as f:  
 lineNum = 1  
 for line in f:  
 if lineNum != 1:  
 text += line + "\n"  
 cursor.insertText(line)  
 cursor.insertText('\n')  
 lineNum += 1  
 stackedWidget.addWidget(text\_edit)  
 listWidget.currentRowChanged.connect(stackedWidget.setCurrentIndex)  
 if type == 1:  
 hbox = QHBoxLayout()  
 hbox.addWidget(listWidget)  
 hbox.addLayout(stackedWidget)  
 vbox.addLayout(hbox)

* 功能描述： 该函数用于在GUI中创建一个包含标注库和历史记录的列表视图，并将对应的文本内容显示在堆叠窗口中。它会读取指定目录下的文件，并将文件名和内容分别添加到列表和堆叠窗口中，同时实现列表和堆叠窗口之间的交互。
* 实现流程： 检查传入的listWidget和stackedWidget是否为None，如果为None，则创建一个新的QListWidget和QStackedLayout，并设置它们的属性。 定义标注库和历史记录的目录路径。 遍历标注库目录中的每个文件，读取文件内容，并将其添加到QListWidget中。同时，创建一个QTextEdit控件，设置为只读，并将文件内容插入到QTextEdit中，然后将QTextEdit添加到QStackedLayout中。 遍历历史记录目录中的每个文件，读取文件内容，并将其添加到QListWidget中。同时，创建一个QTextEdit控件，设置为只读，并将文件内容插入到QTextEdit中，然后将QTextEdit添加到QStackedLayout中。 将QListWidget和QStackedLayout添加到传入的vbox布局中，实现列表和堆叠窗口的布局。 连接QListWidget的currentRowChanged信号到QStackedLayout的setCurrentIndex槽，实现列表和堆叠窗口之间的交互。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： QListWidget,QStackedLayout,setObjectName,setMaximumWidth,listdir,filePath,addItem,QTextEdit,setReadOnly,textCursor,insertText,addWidget,connect,QHBoxLayout,addLayout,
* 内部依赖描述：
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。

### updateChords (VirtualKeyboard.py)

* 行号位置：592-638
* 重要性评分：26.70

**代码片段：**

def updateChords(self):  
 chord\_sequences = []  
 for cbox in self.findChildren(QCheckBox):  
 if cbox.isChecked():  
 logging.info(cbox)  
 with open(filePath('labels/') + cbox.text() + '.model', 'r', encoding='utf-8'  
 ) as f:  
 lineNum = 1  
 for line in f:  
 if lineNum != 1:  
 cline = line.split('||')  
 arr\_str = cline[0].split(',,')  
 chord\_sequences.append(arr\_str)  
 chord\_sequences.append([self.ENDING])  
 lineNum += 1  
 if len(chord\_sequences) != 0:  
 logging.info(f"构建马尔科夫链{chord\_sequences}")  
 if self.image\_window:  
 self.image\_window.set\_chord\_sequences(chord\_sequences)  
 else:  
 if self.image\_window:  
 self.image\_window.set\_chord\_sequences([])  
  
 order = 1 if self.radio\_btn1.isChecked() else 2 if self.radio\_btn2.isChecked() else 3  
  
 if self.QUEUE.length() >= order and len(chord\_sequences) > 0:  
 logging.info(f"阶数: {order}")  
 if chord\_sequences != self.current\_chord\_sequences or self.last\_order is None or self.last\_order != order:  
 self.current\_chord\_sequences = chord\_sequences  
 self.predictor = ChordPredictor(chord\_sequences, order)  
 self.last\_order = order  
 logging.info(f"初始化预测器，阶数：{order}")  
 if order == 3:  
 current\_chords = [self.QUEUE.index(self.QUEUE.length() - 3), self.QUEUE.index(self.QUEUE.length() - 2),  
 self.QUEUE.index(self.QUEUE.length() - 1)]  
 else:  
 current\_chords = [self.QUEUE.index(self.QUEUE.length() - 2), self.QUEUE.index(self.QUEUE.length() - 1)]  
 predictions = self.predictor.predict\_chord(current\_chords)  
  
 check\_sequence = []  
 for sequence in chord\_sequences:  
 check\_sequence += sequence  
 logging.info(f'预测结果: {predictions}')  
 self.next.setText(  
 "预测下一个和弦：(<和弦名称>,<匹配比率%>) " + str(predictions))  
 else:  
 self.next.setText("预测下一个和弦: ")

* 功能描述： 该函数用于更新和弦序列，并根据当前和弦预测下一个和弦。它会读取选中的和弦模型文件，构建马尔科夫链，并根据用户选择的阶数进行预测。预测结果会显示在界面上。
* 实现流程： 遍历所有选中的QCheckBox，检查是否被选中。 如果选中，读取对应的和弦模型文件，构建马尔科夫链。 根据用户选择的阶数，从队列中获取当前和弦状态。 使用ChordPredictor预测下一个和弦，并过滤出概率大于阈值的和弦。 更新图像显示新的和弦序列。 显示预测结果在界面上。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： findChildren,isChecked,info,filePath,text,set\_chord\_sequences,ChordPredictor,index,predict\_chord,setText,
* 内部依赖描述：
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。
  + set\_chord\_sequences: 该函数用于设置新的和弦序列，并在和弦序列发生变化时更新图像。
  + index: 该函数用于从数组中获取指定索引位置的元素。
  + predict\_chord: 该函数用于预测音乐中的下一个和弦。它通过分析当前和弦序列来确定下一个和弦，并计算该和弦的概率。

### pressingEvent (VirtualKeyboard.py)

* 行号位置：466-505
* 重要性评分：22.00

**代码片段：**

def pressingEvent(self):  
 # 获取正在按下的所有琴键  
 i = 0  
 pressingCache = []  
 pressing = []  
 for isPressed in self.key\_pressed:  
 if isPressed:  
 pressingCache.append({'index': i, 'name': self.values[i]})  
 pressing.append(self.values[i])  
 i += 1  
 if 3 <= len(pressing) <= 5:  
 logging.info(f"正在按下的所有琴键：{pressing}")  
 detectElement = musicUtils.detectElement(pressing)  
 chord\_name = detectElement.getNormalChord()  
 item = QLabel('当前和弦： ' + chord\_name)  
 self.chords.setText(item.text())  
 if len(pressing) > self.PRE\_COUNT:  
 self.PRE\_CHORD = (chord\_name, pressingCache)  
 logging.info(self.QUEUE.array)  
 elif len(pressing) == 0:  
 item = QLabel('当前和弦： ')  
 self.chords.setText(item.text())  
 if self.PRE\_CHORD is not None:  
 chord\_name, pressing\_notes = self.PRE\_CHORD  
 if chord\_name != self.QUEUE.last():  
 if self.QUEUE.length() < self.MAX\_QUEUE:  
 logging.info("push")  
 self.QUEUE.push(chord\_name)  
 else:  
 logging.info("pop")  
 self.QUEUE.pop()  
 self.QUEUE.push(chord\_name)  
 # 将 pressing（当前按下的音符）与 self.PRE\_CHORD（当前和弦）的映射关系缓存到本地  
 # 启动异步线程进行文件写入  
 worker = FileWriteWorker(chord\_name, pressing\_notes)  
 QThreadPool.globalInstance().start(worker)  
  
 else:  
 logging.info(f"重复和弦{self.QUEUE.last()}")  
 self.PRE\_COUNT = len(pressing)

* 功能描述： 该函数用于检测当前按下的琴键，并根据按下的琴键数量和音符组合来确定当前的和弦。如果按下的琴键数量在3到5之间，它会调用音乐工具的检测函数来识别和弦，并更新UI显示当前和弦。如果按下的琴键数量为0，它会检查是否有之前的和弦记录，并根据记录的和弦与当前和弦是否相同来决定是否更新队列。如果和弦不同，它会根据队列的长度决定是添加新和弦还是替换旧和弦，并启动异步线程将和弦和对应的音符写入本地文件。
* 实现流程： 获取正在按下的所有琴键，并记录它们的索引和名称。 检查按下的琴键数量是否在3到5之间，如果是，则调用音乐工具的检测函数来识别和弦，并更新UI显示当前和弦。 如果按下的琴键数量为0，检查是否有之前的和弦记录。 如果和弦不同，根据队列的长度决定是添加新和弦还是替换旧和弦，并启动异步线程将和弦和对应的音符写入本地文件。 更新按下的琴键数量的记录。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： info,detectElement,getNormalChord,QLabel,setText,text,last,FileWriteWorker,globalInstance,start,
* 内部依赖描述：
  + getNormalChord: 该函数用于检测音乐中的正常和弦。
  + last: 该函数用于获取数组的最后一个元素。如果数组长度不为0，则返回数组的最后一个元素；否则，不执行任何操作。

### reShaderLabels (VirtualKeyboard.py)

* 行号位置：311-335
* 重要性评分：20.50

**代码片段：**

def reShaderLabels(self):  
 # 清空QHBoxLayout  
 for ch in self.scroll\_content.children():  
 ch.deleteLater()  
  
 self.labelBox = QVBoxLayout()  
 self.labelBox.setObjectName("labelO")  
 self.scroll\_content = QWidget(self.scroll\_area)  
 self.scroll\_content.setObjectName("labelContent")  
 self.scroll\_content.setLayout(self.labelBox)  
 self.scroll\_area.setWidget(self.scroll\_content)  
 labelhbox = QHBoxLayout()  
 count = 0 # 记录已添加的多选框数量  
 for file\_name in os.listdir(filePath('labels/')):  
 name, extension = os.path.splitext(file\_name)  
 check\_box = QCheckBox(name, self)  
 check\_box.setText(name)  
 labelhbox.addWidget(check\_box)  
 count += 1  
 if count >= 5: # 超过x个数量，则另起一行继续水平排列  
 self.labelBox.addLayout(labelhbox)  
 labelhbox = QHBoxLayout()  
 count = 0  
 if count > 0: # 处理剩余不到x个的多选框  
 self.labelBox.addLayout(labelhbox)

* 功能描述： 该函数用于重新设置标签的布局，清空现有的布局并重新添加多选框，每行最多显示5个多选框。
* 实现流程： 清空QHBoxLayout，删除所有子控件。 创建一个新的QVBoxLayout作为主布局，并设置对象名称。 创建一个新的QWidget作为内容容器，并设置对象名称和布局。 将内容容器设置为滚动区域的部件。 遍历指定目录下的所有文件，创建QCheckBox控件并添加到水平布局中。 每添加5个多选框后，将水平布局添加到主布局中，并创建一个新的水平布局。 处理剩余的多选框，确保它们也被添加到主布局中。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： children,deleteLater,QVBoxLayout,setObjectName,QWidget,setLayout,setWidget,QHBoxLayout,listdir,filePath,splitext,QCheckBox,setText,addWidget,addLayout,
* 内部依赖描述：
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。

### **init** (VirtualKeyboard.py)

* 行号位置：75-97
* 重要性评分：19.30

**代码片段：**

def \_\_init\_\_(self, NoneMIDI=False):  
 super().\_\_init\_\_()  
 sys.stdout.flush()  
 self.NoneMIDI = NoneMIDI  
 # 添加布局  
 self.vbox = QVBoxLayout()  
 self.initUI()  
 # 当前和弦展示项  
 self.chords = self.findChild(QLabel, "chords")  
 self.next = self.findChild(QLabel, "next")  
 self.listWidget = self.findChild(QListWidget, "listO")  
 self.stackedWidget = self.findChild(QStackedLayout, "stackO")  
 self.labelBox = self.findChild(QVBoxLayout, "labelO")  
 self.scroll\_content = self.findChild(QWidget, "labelContent")  
 self.scroll\_area = self.findChild(QScrollArea, "scroll")  
  
 self.timer = QTimer(self)  
 self.timer.timeout.connect(self.updateChords)  
  
 self.updateTimer = QTimer(self)  
 self.updateTimer.timeout.connect(self.updateButtonColor)  
 self.updateTimer.timeout.connect(self.updateMIDI)  
 self.clearFocus()

* 功能描述： 该类用于初始化一个支持MIDI输入和键盘映射的系统，主要用于和弦序列预测，并支持情绪和风格标签作为上下文。它设置了一个窗口标题和大小，并初始化了一些变量来控制图像更新。类中还设置了标签来显示图像，并通过布局管理器将标签添加到窗口中。
* 实现流程： 初始化类，设置是否支持MIDI输入。 调用父类的初始化方法。 刷新标准输出。 设置是否支持MIDI输入的变量。 创建垂直布局。 初始化用户界面。 查找并初始化各个UI组件，如标签、列表、堆叠布局等。 创建定时器，用于更新和弦显示。 创建另一个定时器，用于更新按钮颜色和MIDI输入。 清除焦点
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： **init**,flush,QVBoxLayout,initUI,findChild,QTimer,connect,clearFocus,
* 内部依赖描述：
  + **init**: 该类用于初始化一个支持MIDI输入和键盘映射的系统，主要用于和弦序列预测，并支持情绪和风格标签作为上下文。它设置了一个窗口标题和大小，并初始化了一些变量来控制图像更新。类中还设置了标签来显示图像，并通过布局管理器将标签添加到窗口中。
  + initUI: 该函数用于初始化用户界面，设置窗口大小和标题，根据是否使用MIDI键盘显示不同的标题，并创建各种控件和布局，包括按键、标签、单选按钮、多选框、文本框和按钮等。这些控件用于处理音乐创作、和弦识别、预测和保存等功能。

### baum\_welch (service/ChordHMMV2.py)

* 行号位置：141-200
* 重要性评分：19.00

**代码片段：**

def baum\_welch(self, max\_iters=10):  
 """  
 Baum-Welch算法，用于训练HMM参数，考虑上下文依赖。  
 :param max\_iters: 最大迭代次数  
 """  
 N = self.n\_states  
 M = self.n\_chords  
  
 sequences = [self.chords\_to\_indices(seq) for seq in self.chord\_sequences]  
  
 for iteration in range(max\_iters):  
 A\_num = np.zeros((N, N))  
 A\_den = np.zeros(N)  
 B\_num = np.zeros((N, M, self.n\_contexts))  
 B\_den = np.zeros((N, self.n\_contexts))  
  
 pi\_sum = np.zeros(N)  
  
 for O, emotions, style in zip(sequences, self.emotion\_labels, self.style\_labels):  
 context\_idx = self.context\_to\_index(emotions, style)  
 if context\_idx == -1:  
 continue # 跳过未知上下文  
  
 T = len(O)  
 alpha = self.forward(O, context\_idx, self.A, self.B, self.pi)  
 beta = self.backward(O, context\_idx, self.A, self.B)  
 xi = np.zeros((T - 1, N, N))  
 gamma = np.zeros((T, N))  
  
 # 计算xi和gamma  
 for t in range(T - 1):  
 denom = np.sum(alpha[t] \* beta[t])  
 for i in range(N):  
 gamma[t, i] = alpha[t, i] \* beta[t, i] / denom  
 b = self.B[:, O[t + 1], context\_idx]  
 xi[t, i, :] = alpha[t, i] \* self.A[i, :] \* b \* beta[t + 1] / denom  
  
 gamma[T - 1] = alpha[T - 1] \* beta[T - 1] / np.sum(alpha[T - 1] \* beta[T - 1])  
  
 # 累积A和B的更新量  
 A\_num += np.sum(xi, axis=0)  
 A\_den += np.sum(gamma[:-1], axis=0)  
 for t in range(T):  
 B\_num[:, O[t], context\_idx] += gamma[t]  
 B\_den[:, context\_idx] += np.sum(gamma, axis=0)  
  
 pi\_sum += gamma[0]  
  
 # 更新模型参数  
 self.A = A\_num / A\_den[:, None]  
 # 防止除零  
 self.A = np.nan\_to\_num(self.A)  
 self.B = B\_num / B\_den[:, None, :]  
 self.B = np.nan\_to\_num(self.B)  
 self.pi = pi\_sum / len(sequences)  
  
 # 防止数值问题，进行归一化  
 self.A /= self.A.sum(axis=1, keepdims=True)  
 self.B /= self.B.sum(axis=1, keepdims=True)  
 self.pi /= self.pi.sum()

* 功能描述： 该函数实现了Baum-Welch算法，用于训练隐马尔可夫模型（HMM）的参数，考虑上下文依赖。通过迭代更新状态转移矩阵A、观测概率矩阵B和初始状态概率向量pi，以最大化观察序列的似然性。
* 实现流程： 初始化状态转移矩阵A、观测概率矩阵B和初始状态概率向量pi。 将和弦序列转换为对应的索引序列。 对于每个迭代，初始化A\_num、A\_den、B\_num、B\_den和pi\_sum。 遍历每个观察序列、情绪标签和风格标签，计算上下文索引。 使用前向算法计算状态概率矩阵alpha。 使用后向算法计算贝塔值矩阵beta。 计算xi和gamma，用于更新A和B的更新量。 累积A和B的更新量，并更新pi\_sum。 更新模型参数A、B和pi，防止除零和数值问题，进行归一化。 重复上述步骤，直到达到最大迭代次数或收敛。
* 引入包： time,pickle,utils.filePath,
* 调用： chords\_to\_indices,zeros,context\_to\_index,forward,backward,nan\_to\_num,
* 内部依赖描述：
  + chords\_to\_indices: 将和弦序列转换为对应的索引序列。
  + context\_to\_index: 该函数用于将情绪标签和风格标签转换为上下文索引。它首先对情绪标签进行排序并转换为元组，然后与风格标签一起组成上下文元组。最后，通过上下文元组在self.context\_index字典中查找对应的索引，如果存在则返回索引，否则返回-1。
  + forward: 前向算法用于计算给定观察序列在隐马尔可夫模型中的概率，通过计算Alpha矩阵来实现。
  + backward: 后向算法用于计算贝塔值，贝塔值在隐马尔可夫模型中用于计算给定观察序列下每个状态在每个时间步的概率。

### onLabel (VirtualKeyboard.py)

* 行号位置：367-393
* 重要性评分：18.70

**代码片段：**

def onLabel(self):  
 if self.edit\_2.text() != "":  
 directory = filePath("labels/")  
 filenames = self.edit\_2.text().split(',')  
 for filename in filenames:  
 filename += ".model"  
 logging.info(filename)  
 file\_path = os.path.join(directory, filename)  
 if filename != "":  
 if os.path.exists(file\_path):  
 with open(os.path.join(directory, filename), 'a', encoding='utf-8') as f:  
 f.write("\n")  
 csv\_str = ',,'.join(self.QUEUE.array)  
 f.write(csv\_str + "||" + self.edit\_3.text())  
  
 else:  
 with open(os.path.join(directory, filename), 'w', encoding='utf-8') as f:  
 f.write(self.edit\_2.text())  
 f.write("\n")  
 csv\_str = ',,'.join(self.QUEUE.array)  
 f.write(csv\_str + "||" + self.edit\_3.text())  
 # 刷新list  
 self.reShaderLists()  
 # 刷新label  
 self.reShaderLabels()  
 else:  
 logging.info("填写为空")

* 功能描述： 该函数用于处理标签文件的创建和更新。当输入框edit\_2不为空时，它会根据输入的文件名列表创建或更新相应的.model文件，并将QUEUE数组的内容以特定格式写入文件。如果文件已存在，则追加内容；如果不存在，则创建新文件并写入初始内容。最后，刷新着色器列表和标签。
* 实现流程： 检查edit\_2文本是否为空，如果不为空则继续执行，否则记录日志并返回。 构建文件路径，使用filePath函数将’labels/‘目录与当前脚本所在目录的上两级目录拼接。 将edit\_2文本按逗号分割，获取文件名列表。 遍历文件名列表，为每个文件名添加’.model’后缀，并记录日志。 检查文件是否存在，如果存在则以追加模式打开文件，否则以写入模式打开文件。 将QUEUE数组的内容以特定格式写入文件，格式为’,,‘.join(self.QUEUE.array) +’||’ + self.edit\_3.text()。 关闭文件。 调用reShaderLists函数刷新着色器列表。 调用reShaderLabels函数刷新标签。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： text,filePath,info,exists,write,reShaderLists,reShaderLabels,
* 内部依赖描述：
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。
  + reShaderLists: 该函数用于清空列表小部件和堆叠小部件中的所有内容，并重新加载新的着色器列表。
  + reShaderLabels: 该函数用于重新设置标签的布局，清空现有的布局并重新添加多选框，每行最多显示5个多选框。

### update\_image (MatrixView.py)

* 行号位置：37-62
* 重要性评分：18.60

**代码片段：**

def update\_image(self):  
 """Update the displayed image based on the current chord sequences."""  
 if not self.needs\_update or not self.chord\_sequences:  
 return  
 # Create a matplotlib figure  
 fig = plt.figure(figsize=(5, 4))  
 ax = fig.add\_subplot(111)  
 # Create the predictor and plot the transition matrix  
 self.predictor = ChordPredictor(self.chord\_sequences, order=1)  
 cax = ax.imshow(self.predictor.transitions, cmap='hot', interpolation='nearest')  
 fig.colorbar(cax)  
 ax.set\_title("Transition Matrix Heatmap")  
 ax.set\_xlabel("Next Chord Index")  
 ax.set\_ylabel("Current State Index")  
 # Render the figure to a buffer  
 buf = io.BytesIO()  
 fig.savefig(buf, format='png')  
 buf.seek(0)  
 plt.close(fig)  
 # Load the image from the buffer and set it to the label  
 img = QImage.fromData(buf.getvalue())  
 pixmap = QPixmap.fromImage(img)  
 self.label.setPixmap(pixmap)  
 buf.close()  
 self.update()  
 self.needs\_update = False

* 功能描述： 该函数用于根据当前和预测的和弦序列更新显示的图像，通过绘制和弦转换矩阵的热图来实现。
* 实现流程： 检查是否需要更新或和弦序列是否存在，如果不需要则直接返回。 创建一个matplotlib图形对象，并添加一个子图。 使用当前和弦序列创建一个和弦预测器，并绘制其转换矩阵的热图。 将图形保存到一个内存缓冲区中，并关闭图形对象。 从缓冲区加载图像数据，并将其转换为QImage和QPixmap。 将QPixmap设置为标签的图像，并更新标签。 关闭内存缓冲区，并将更新标志设置为False
* 引入包： sys,PyQt5.QtWidgets,PyQt5.QtCore,PyQt5.QtGui,service.numpyMarkov,io,
* 调用： figure,add\_subplot,ChordPredictor,imshow,colorbar,set\_title,set\_xlabel,set\_ylabel,BytesIO,savefig,seek,fromData,getvalue,fromImage,setPixmap,update,
* 内部依赖描述：

### onRecord (VirtualKeyboard.py)

* 行号位置：345-365
* 重要性评分：16.10

**代码片段：**

def onRecord(self):  
 if self.edit\_1.text() != "":  
 directory = filePath("records/")  
 filename = self.edit\_1.text() + ".model"  
 file\_path = os.path.join(directory, filename)  
 if os.path.exists(file\_path):  
 with open(os.path.join(directory, filename), 'a', encoding='utf-8') as f:  
 f.write("\n")  
 csv\_str = ',,'.join(self.QUEUE.array)  
 f.write(csv\_str + "||" + self.edit\_3.text())  
 else:  
 with open(os.path.join(directory, filename), 'w', encoding='utf-8') as f:  
 f.write(self.edit\_2.text())  
 f.write("\n")  
 csv\_str = ',,'.join(self.QUEUE.array)  
 f.write(csv\_str + "||" + self.edit\_3.text())  
  
 # 刷新list  
 self.reShaderLists()  
 else:  
 logging.info("填写为空")

* 功能描述： 该函数用于处理记录操作，当编辑框1不为空时，根据编辑框1的文本创建或追加文件，并将队列数组和编辑框3的文本写入文件。如果文件已存在，则追加内容；否则，创建新文件并写入初始内容。最后，刷新列表以显示新的着色器列表。
* 实现流程： 检查编辑框1的文本是否为空，如果不为空则继续执行，否则记录日志并返回。 构建文件路径，使用filePath函数将记录目录与当前脚本所在目录的上两级目录拼接。 根据文件是否存在，决定是追加内容还是创建新文件。 将队列数组和编辑框3的文本以特定格式写入文件。 调用reShaderLists函数刷新列表，显示新的着色器列表。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： text,filePath,exists,write,reShaderLists,info,
* 内部依赖描述：
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。
  + reShaderLists: 该函数用于清空列表小部件和堆叠小部件中的所有内容，并重新加载新的着色器列表。

### baum\_welch (service/ChordHMM.py)

* 行号位置：96-146
* 重要性评分：15.10

**代码片段：**

def baum\_welch(self, max\_iters=10):  
 """  
 Baum-Welch算法，用于训练HMM参数。  
 :param max\_iters: 最大迭代次数  
 """  
 N = self.n\_states  
 M = self.n\_chords  
 sequences = [self.chords\_to\_indices(seq) for seq in self.chord\_sequences]  
  
 for iteration in range(max\_iters):  
 A\_num = np.zeros((N, N))  
 A\_den = np.zeros(N)  
 B\_num = np.zeros((N, M))  
 B\_den = np.zeros(N)  
  
 pi\_sum = np.zeros(N)  
  
 for O in sequences:  
 T = len(O)  
 alpha = self.forward(O, self.A, self.B, self.pi)  
 beta = self.backward(O, self.A, self.B)  
 xi = np.zeros((T - 1, N, N))  
 gamma = np.zeros((T, N))  
  
 # 计算xi和gamma  
 for t in range(T - 1):  
 denom = np.sum(alpha[t] \* beta[t])  
 for i in range(N):  
 gamma[t, i] = alpha[t, i] \* beta[t, i] / denom  
 xi[t, i, :] = alpha[t, i] \* self.A[i, :] \* self.B[:, O[t + 1]] \* beta[t + 1] / denom  
  
 gamma[T - 1] = alpha[T - 1] \* beta[T - 1] / np.sum(alpha[T - 1] \* beta[T - 1])  
  
 # 累积A和B的更新量  
 A\_num += np.sum(xi, axis=0)  
 A\_den += np.sum(gamma[:-1], axis=0)  
 for t in range(T):  
 B\_num[:, O[t]] += gamma[t]  
 B\_den += np.sum(gamma, axis=0)  
  
 pi\_sum += gamma[0]  
  
 # 更新模型参数  
 self.A = A\_num / A\_den[:, None]  
 self.B = B\_num / B\_den[:, None]  
 self.pi = pi\_sum / len(sequences)  
  
 # 防止数值问题，进行归一化  
 self.A /= self.A.sum(axis=1, keepdims=True)  
 self.B /= self.B.sum(axis=1, keepdims=True)  
 self.pi /= self.pi.sum()

* 功能描述： 该函数实现了Baum-Welch算法，用于训练隐马尔可夫模型（HMM）的参数。通过迭代更新状态转移矩阵A、观测概率矩阵B和初始状态概率向量pi，使得模型能够更好地拟合给定的和弦序列。
* 实现流程： 初始化模型参数A、B和pi。 将和弦序列转换为对应的索引序列。 进行最大迭代次数的循环： 在每次迭代中，计算前向概率矩阵alpha和后向概率矩阵beta。 计算xi和gamma，用于更新A、B和pi的更新量。 累积A、B和pi的更新量。 更新模型参数A、B和pi。 防止数值问题，进行归一化处理。 输出最终训练好的模型参数。
* 引入包： pickle,utils.filePath,
* 调用： chords\_to\_indices,zeros,forward,backward,
* 内部依赖描述：
  + chords\_to\_indices: 将和弦序列转换为对应的索引序列。
  + forward: 前向算法用于计算给定观察序列在隐马尔可夫模型中的概率，通过计算Alpha矩阵来实现。
  + backward: 后向算法用于计算贝塔值，贝塔值在隐马尔可夫模型中用于计算给定观察序列下每个状态在每个时间步的概率。

### updateMIDI (VirtualKeyboard.py)

* 行号位置：442-463
* 重要性评分：14.20

**代码片段：**

def updateMIDI(self):  
 # 渲染实时和弦序列  
 if self.QUEUE.length() > 10:  
 for i in range(0, self.MAX\_QUEUE):  
 label = self.findChild(QPushButton, "ch" + str(i))  
 font = QFont('Calibri', 16)  
 label.setFont(font)  
 # self.reShaderButton(label)  
 label.setText(self.QUEUE.array[i])  
 else:  
 for i in range(0, self.QUEUE.length()):  
 label = self.findChild(QPushButton, "ch" + str(i))  
 font = QFont('Calibri', 16)  
 label.setFont(font)  
 # self.reShaderButton(label)  
 label.setText(self.QUEUE.array[i])  
 for j in range(self.QUEUE.length(), self.MAX\_QUEUE):  
 label = self.findChild(QPushButton, "ch" + str(j))  
 font = QFont('Calibri', 16)  
 label.setFont(font)  
 # self.reShaderButton(label)  
 label.setText("")

* 功能描述： 该函数用于更新MIDI界面的和弦序列显示。当队列长度超过10时，只更新前10个和弦；否则，更新所有队列中的和弦，并将剩余的和弦显示为空字符串。
* 实现流程： 检查队列长度是否大于10。 如果队列长度大于10，遍历前10个队列项，更新对应的按钮文本和字体。 如果队列长度小于或等于10，遍历所有队列项，更新对应的按钮文本和字体，并将剩余的按钮文本设置为空字符串。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： findChild,QFont,setFont,setText,
* 内部依赖描述：

### keyPressEvent (VirtualKeyboard.py)

* 行号位置：573-589
* 重要性评分：13.70

**代码片段：**

def keyPressEvent(self, event):  
 self.setFocusPolicy(Qt.StrongFocus)  
 logging.info(f"Key pressed: {event.key()}")  
 if str(event.key()) == '96' or str(event.key()) == '126' or str(event.key()) == '183':  
 if self.NoneMIDI:  
 # 切换监听模式  
 self.listening\_enabled = not self.listening\_enabled  
 if self.listening\_enabled:  
 self.status\_label.setText("<~键>控制当前监听状态：已开启")  
 self.status\_label.setStyleSheet("font-size: 12px; color: green;")  
 else:  
 self.status\_label.setText("<~键>控制当前监听状态：已关闭")  
 self.status\_label.setStyleSheet("font-size: 12px; color: red;")  
 logging.info(f"监听模式：{'开启' if self.listening\_enabled else '关闭'}")  
 else:  
 # 调用父类的按键处理  
 super().keyPressEvent(event)

* 功能描述： 该函数处理键盘按下事件，当按下特定键（96、126、183）时，切换监听模式，并更新状态标签的显示和样式。
* 实现流程： 设置焦点策略为强焦点。 记录按键事件的键值。 检查按键是否为96、126或183。 如果按键是上述之一，检查当前监听模式是否为NoneMIDI。 切换监听模式，并根据模式更新状态标签的文本和样式。 记录监听模式的切换状态。 如果按键不是上述之一，调用父类的按键处理方法。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： setFocusPolicy,info,key,setText,setStyleSheet,keyPressEvent,
* 内部依赖描述：
  + keyPressEvent: 该函数处理键盘按下事件，当按下特定键（96、126、183）时，切换监听模式，并更新状态标签的显示和样式。

### run (utils/FileWorker.py)

* 行号位置：25-65
* 重要性评分：11.10

**代码片段：**

def run(self):  
 mapping\_file = filePath('models/pressing\_chord\_mappings.cache')  
 # 创建文件夹如果不存在  
 os.makedirs(os.path.dirname(mapping\_file), exist\_ok=True)  
  
 print(f"{self.chord\_name} and {self.pressing\_notes}")  
  
 # 将按下的音符转换为音符名称和音符索引  
 root\_note = self.pressing\_notes[0]['name'] # 第一个音符为根音  
 note\_indices = [str(note['index']) for note in self.pressing\_notes]  
 note\_indices\_str = ','.join(note\_indices)  
 pressing\_str = f"{root\_note}@{note\_indices\_str}"  
  
 # 读取现有的映射文件，检查去重  
 lines = []  
 if os.path.exists(mapping\_file):  
 with open(mapping\_file, 'r', encoding='utf-8') as f:  
 lines = f.readlines()  
  
 # 查找是否已经存在相同的和弦名称  
 found\_chord = False  
 for i, line in enumerate(lines):  
 line = line.strip()  
 if not line:  
 continue  
 parts = line.split('::')  
 if parts[0] == self.chord\_name:  
 found\_chord = True  
 # 检查是否已经存在相同的根音音符  
 if pressing\_str not in parts[1:]:  
 lines[i] = line + '::' + pressing\_str + '\n'  
 break  
  
 # 如果没有找到相同的和弦名称，添加新的记录  
 if not found\_chord:  
 new\_line = f"{self.chord\_name}::{pressing\_str}\n"  
 lines.append(new\_line)  
  
 # 写回文件  
 with open(mapping\_file, 'w', encoding='utf-8') as f:  
 f.writelines(lines)

* 功能描述： 该函数用于管理和弦名称及其对应的按下音符的映射关系。它首先构建映射文件的路径，然后检查文件是否存在。如果存在，它会读取文件内容并检查是否已经存在相同的和弦名称。如果存在相同的和弦名称，它会检查是否已经存在相同的根音音符，如果不存在，则添加新的音符。如果不存在相同的和弦名称，则添加新的记录。最后，它将更新后的映射写回文件。
* 实现流程： 构建映射文件的路径。 检查映射文件是否存在。 如果文件存在，读取文件内容并检查是否已经存在相同的和弦名称。 如果存在相同的和弦名称，检查是否已经存在相同的根音音符，如果不存在，则添加新的音符。 如果不存在相同的和弦名称，则添加新的记录。 将更新后的映射写回文件。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,utils.filePath,
* 调用： filePath,makedirs,dirname,exists,readlines,strip,writelines,
* 内部依赖描述：
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。

### reShaderButton (VirtualKeyboard.py)

* 行号位置：522-542
* 重要性评分：11.10

**代码片段：**

def reShaderButton(self, button):  
 # 获取按钮的最大宽度  
 max\_width = button.width()  
  
 # 获取按钮的字体  
 font = button.font()  
  
 # 创建一个QFontMetrics对象，用于计算不同字体/大小下的文本大小  
 metrics = QFontMetrics(font)  
  
 # 计算当前字体下文本的宽度  
 text\_width = metrics.width(button.text())  
  
 logging.info(f"按钮宽度{max\_width}和文本宽度{text\_width}")  
  
 # 如果文本太长，则根据按钮宽度自动调整字体大小，使文本填充按钮  
 if text\_width > max\_width:  
 # 计算调整后的字体大小  
 font\_size = max\_width \* font.pointSize() // text\_width  
 font.setPointSize(font\_size)  
 button.setFont(font)

* 功能描述： 该函数用于调整按钮上的文本字体大小，以确保文本在按钮内完整显示。如果文本宽度超过按钮宽度，则会自动调整字体大小，使文本适应按钮的宽度。
* 实现流程： 获取按钮的最大宽度。 获取按钮的当前字体。 创建一个QFontMetrics对象，用于计算不同字体/大小下的文本大小。 计算当前字体下文本的宽度。 记录按钮宽度和文本宽度的日志信息。 如果文本宽度超过按钮宽度，则计算调整后的字体大小。 设置调整后的字体大小到按钮上。 更新按钮的字体属性。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： width,font,QFontMetrics,text,info,pointSize,setPointSize,setFont,
* 内部依赖描述：

### **init** (MatrixView.py)

* 行号位置：11-28
* 重要性评分：10.80

**代码片段：**

def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("概率转移矩阵的和弦分布图")  
 self.setGeometry(200, 200, 500, 400)  
  
 # Initialize variables  
 self.chord\_sequences = []  
 self.needs\_update = False # Flag to control image updates  
  
 # Set up the label to display the image  
 self.label = QLabel(self)  
 self.layout = QVBoxLayout()  
 self.layout.addWidget(self.label)  
  
 # Since QMainWindow cannot have a layout directly, set a central widget  
 widget = QWidget()  
 widget.setLayout(self.layout)  
 self.setCentralWidget(widget)

* 功能描述： 该类用于初始化一个支持MIDI输入和键盘映射的系统，主要用于和弦序列预测，并支持情绪和风格标签作为上下文。它设置了一个窗口标题和大小，并初始化了一些变量来控制图像更新。类中还设置了标签来显示图像，并通过布局管理器将标签添加到窗口中。
* 实现流程： 初始化窗口标题和大小。 初始化变量，包括和弦序列列表和一个标志位，用于控制图像更新。 设置一个标签来显示图像，并将其添加到布局管理器中。 创建一个中心部件，并将布局管理器设置为中心部件的布局。 将中心部件设置为主窗口的中心部件。
* 引入包： sys,PyQt5.QtWidgets,PyQt5.QtCore,PyQt5.QtGui,service.numpyMarkov,io,
* 调用： **init**,setWindowTitle,setGeometry,QLabel,QVBoxLayout,addWidget,QWidget,setLayout,setCentralWidget,
* 内部依赖描述：
  + **init**: 该类用于初始化一个支持MIDI输入和键盘映射的系统，主要用于和弦序列预测，并支持情绪和风格标签作为上下文。它设置了一个窗口标题和大小，并初始化了一些变量来控制图像更新。类中还设置了标签来显示图像，并通过布局管理器将标签添加到窗口中。

### **init** (service/soundNoise.py)

* 行号位置：10-34
* 重要性评分：10.50
* 功能描述： 该类用于初始化一个支持MIDI输入和键盘映射的系统，主要用于和弦序列预测，并支持情绪和风格标签作为上下文。它会根据操作系统选择合适的音频驱动，并初始化Fluidsynth库，加载SoundFont文件以提供音乐音色。
* 实现流程： 初始化Fluidsynth并加载SoundFont 检测操作系统并选择合适的音频驱动 初始化Fluidsynth，并启动合适的音频驱动 加载SoundFont文件 选择钢琴音色
* 引入包： platform,fluidsynth,utils.filePath,logging,
* 调用： **init**,info,Synth,start,filePath,sfload,program\_select,
* 内部依赖描述：
  + **init**: 该类用于初始化一个支持MIDI输入和键盘映射的系统，主要用于和弦序列预测，并支持情绪和风格标签作为上下文。它设置了一个窗口标题和大小，并初始化了一些变量来控制图像更新。类中还设置了标签来显示图像，并通过布局管理器将标签添加到窗口中。
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。

### delete\_chord (service/ChordVectorFaiss.py)

* 行号位置：117-143
* 重要性评分：8.70
* 功能描述： 该函数用于从索引中删除指定的和弦。它首先遍历当前的索引和映射，将不匹配指定和弦名称的向量和相关信息保存到新的列表和映射中。然后，它重建索引，重新添加剩余的向量，并保存新的索引和映射。
* 实现流程： 遍历当前的索引和映射，将不匹配指定和弦名称的向量和相关信息保存到新的列表和映射中。 重建索引，使用新的向量列表重新添加向量。 保存新的索引和映射到指定路径。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： items,midi\_notes\_to\_vector,IndexFlatL2,add,array,save\_index,
* 内部依赖描述：
  + midi\_notes\_to\_vector: 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。
  + save\_index: 该函数用于保存索引和映射。它将索引写入指定路径，并将映射对象序列化后保存到另一个指定路径。

### visualize\_emission\_matrix (service/ChordHMMV2.py)

* 行号位置：288-302
* 重要性评分：8.50
* 功能描述： 该函数用于可视化发射矩阵B（仅针对单个上下文）。它首先检查是否存在上下文数据，如果不存在则打印提示信息并返回。如果存在上下文数据，则使用matplotlib库创建一个热图，展示发射矩阵B的第一个上下文的数据，热图的x轴表示音阶索引，y轴表示隐藏状态索引，并显示颜色条以表示数值大小。
* 实现流程： 检查发射矩阵B的上下文数量是否为0，如果是，则打印提示信息并返回。 创建一个新的图形，设置大小为10x8英寸。 使用matplotlib的imshow函数显示发射矩阵B的第一个上下文的数据，使用’hot’颜色映射，并设置插值为’nearest’。 添加颜色条以显示数值大小。 设置图形的标题为’Emission Matrix Heatmap (Context 0)‘。 设置x轴标签为’Chord Index’，y轴标签为’Hidden State Index’。 显示图形
* 引入包： time,pickle,utils.filePath,
* 调用： figure,imshow,colorbar,title,xlabel,ylabel,show,
* 内部依赖描述：

### add\_chord (service/ChordVectorFaiss.py)

* 行号位置：42-85
* 重要性评分：8.40
* 功能描述： 该函数用于将和弦名称和按键字符串添加到索引中，并将其转换为 One-Hot 编码的向量表示。如果 force\_insert 为 False，则在插入前检查是否已经存在相同的和弦名称和按键字符串，避免重复插入。
* 实现流程： 检查 force\_insert 参数，如果为 False，则遍历 id\_mapping 检查是否存在相同的和弦名称和按键字符串，如果存在则跳过插入。 解析 pressing\_str，提取根音和 MIDI 音符序列。 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。 获取当前索引的总数，作为新向量的 ID。 将新向量添加到索引中。 更新 id\_mapping，记录新添加的和弦名称、按键字符串和 MIDI 音符序列。 更新 next\_id，为下一个向量分配新的 ID。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： items,midi\_notes\_to\_vector,add,array,
* 内部依赖描述：
  + midi\_notes\_to\_vector: 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。

### visualize\_transition\_matrix (service/numpyMarkov.py)

* 行号位置：97-108
* 重要性评分：8.20
* 功能描述： 该函数用于可视化概率转移矩阵，并将结果以热图的形式输出为图片。
* 实现流程： 创建一个大小为10x8的图形窗口。 使用热图展示转移矩阵，颜色映射为’hot’，插值方法为’nearest’。 添加颜色条以显示颜色与数值的对应关系。 设置图形的标题为’Transition Matrix Heatmap’。 设置X轴标签为’Next Chord Index’，Y轴标签为’Current State Index’。 显示图形窗口，输出热图图片。
* 调用： figure,imshow,colorbar,title,xlabel,ylabel,show,
* 内部依赖描述：

### visualize\_emission\_matrix (service/ChordHMM.py)

* 行号位置：214-224
* 重要性评分：8.10
* 功能描述： 该函数用于可视化发射矩阵B，通过热图展示发射概率，其中横轴表示音程索引，纵轴表示隐藏状态索引。
* 实现流程： 创建一个大小为10x8的图形窗口。 使用plt.imshow函数显示发射矩阵B，使用’hot’颜色映射和’nearest’插值方法。 添加颜色条以显示概率值的范围。 设置图形的标题为’Emission Matrix Heatmap’。 设置横轴标签为’Chord Index’，纵轴标签为’Hidden State Index’。 显示图形窗口以供用户查看。
* 引入包： pickle,utils.filePath,
* 调用： figure,imshow,colorbar,title,xlabel,ylabel,show,
* 内部依赖描述：

### predict\_next\_chords (service/ChordHMMV2.py)

* 行号位置：228-257
* 重要性评分：8.00
* 功能描述： 该函数根据当前和弦序列、情绪标签和风格标签，使用Viterbi算法预测下一个和弦，并返回概率高于阈值的和弦及其匹配概率的列表。
* 实现流程： 将当前和弦序列转换为对应的索引序列。 检查序列中是否存在未知和弦，如果存在则返回None。 将情绪标签和风格标签转换为上下文索引。 如果上下文索引无效，则返回None。 使用Viterbi算法找到最可能的隐藏状态序列。 获取最可能的隐藏状态序列的最后一个状态。 根据最后一个状态和上下文索引获取可能的下一个和弦及其概率。 过滤概率高于阈值的和弦。 按概率降序排序并返回预测的和弦及其匹配概率的列表。
* 引入包： time,pickle,utils.filePath,
* 调用： chords\_to\_indices,context\_to\_index,viterbi,where,sort,
* 内部依赖描述：
  + chords\_to\_indices: 将和弦序列转换为对应的索引序列。
  + context\_to\_index: 该函数用于将情绪标签和风格标签转换为上下文索引。它首先对情绪标签进行排序并转换为元组，然后与风格标签一起组成上下文元组。最后，通过上下文元组在self.context\_index字典中查找对应的索引，如果存在则返回索引，否则返回-1。
  + viterbi: Viterbi算法用于在给定观察序列的情况下，找到最可能的隐藏状态序列。

### viterbi (service/ChordHMMV2.py)

* 行号位置：202-226
* 重要性评分：7.50
* 功能描述： Viterbi算法用于在给定观察序列和上下文索引的情况下，找到最可能的隐藏状态序列。
* 实现流程： 初始化delta和psi数组，用于存储每个时间步和状态的最大概率和前驱状态。 根据初始概率pi和发射概率B，计算第一个时间步的概率delta[0]。 从第二个时间步开始，遍历每个时间步，计算每个状态的最大概率delta[t, j]和前驱状态psi[t, j]。 根据delta数组，从最后一个时间步开始回溯，找到最可能的状态序列。 返回最可能的状态序列。
* 引入包： time,pickle,utils.filePath,
* 调用： zeros,argmax,
* 内部依赖描述：

### viterbi (service/ChordHMM.py)

* 行号位置：148-171
* 重要性评分：7.40
* 功能描述： Viterbi算法用于在给定观察序列的情况下，找到最可能的隐藏状态序列。
* 实现流程： 初始化delta矩阵和psi矩阵，delta矩阵用于存储每个时间步每个状态的最大概率，psi矩阵用于存储每个时间步每个状态的最大概率对应的前一个状态。 设置delta矩阵的第一个时间步为初始概率pi与观察序列第一个观测值对应的发射概率B的乘积。 从第二个时间步开始，遍历每个时间步，对于每个状态，计算从上一个时间步的每个状态转移过来的最大概率，并更新delta矩阵和psi矩阵。 初始化状态序列states，设置最后一个时间步的状态为delta矩阵最后一个时间步的最大概率对应的索引。 从倒数第二个时间步开始，逆序遍历每个时间步，根据psi矩阵找到前一个时间步的状态，并更新状态序列。 返回状态序列states，即为最可能的隐藏状态序列。
* 引入包： pickle,utils.filePath,
* 调用： zeros,argmax,
* 内部依赖描述：

### **init** (service/ChordVectorFaiss.py)

* 行号位置：10-30
* 重要性评分：7.10
* 功能描述： 该类用于管理一个基于 Faiss 的向量索引，支持向量的添加、搜索和加载功能。它通过加载已有的索引和映射文件来初始化索引和映射，如果没有找到这些文件，则创建一个新的索引。
* 实现流程： 初始化索引和映射变量。 检查索引和映射文件是否存在。 如果文件存在，加载索引和映射，并设置下一个可用的 ID。 如果文件不存在，创建一个新的 L2 距离的平面索引，并设置下一个可用的 ID 为 0。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： exists,read\_index,keys,IndexFlatL2,
* 内部依赖描述：

### **init** (service/ChordHMMV2.py)

* 行号位置：11-40
* 重要性评分：7.00
* 功能描述： 该函数用于初始化一个隐马尔可夫模型（HMM），用于和弦序列预测，并支持情绪标签和风格标签作为上下文。它通过为和弦分配唯一索引、构建上下文以及随机初始化HMM参数来完成初始化。
* 实现流程： 为和弦序列中的每个和弦分配一个唯一的索引，并返回一个和弦到索引的映射字典。 构建上下文，通过将情绪标签列表和风格标签组合成元组，并将这些元组添加到集合中，最终返回一个上下文的列表。 随机初始化隐马尔可夫模型（HMM）的参数，包括状态转移矩阵A、发射矩阵B和初始状态概率pi。发射矩阵B是三维的，考虑了上下文依赖。
* 引入包： time,pickle,utils.filePath,
* 调用： index\_chords,items,build\_contexts,initialize\_parameters,
* 内部依赖描述：
  + index\_chords: 该函数用于为和弦序列中的每个唯一和弦分配一个唯一的索引。
  + build\_contexts: 该函数用于构建上下文，通过将情绪标签列表和风格标签组合成元组，并将这些元组添加到集合中，最终返回一个上下文的列表。这样可以确保每个上下文都是唯一的。
  + initialize\_parameters: 该函数用于随机初始化隐马尔可夫模型（HMM）的参数，包括状态转移矩阵A、发射矩阵B和初始状态概率pi。

### generate\_chord\_sequence (service/ChordHMMV2.py)

* 行号位置：259-286
* 重要性评分：6.80
* 功能描述： 该函数使用训练好的HMM模型生成指定长度的和弦序列，并支持情绪和风格标签的输入。它首先将情绪和风格标签转换为上下文索引，然后根据HMM模型的转移概率和观测概率生成和弦序列。
* 实现流程： 接收输入参数：序列长度、情绪标签列表和风格标签。 调用context\_to\_index函数将情绪和风格标签转换为上下文索引。 如果上下文索引无效，打印错误信息并返回None。 初始化状态和观测列表。 随机选择初始状态。 根据HMM模型的转移概率和观测概率，循环生成指定长度的和弦序列。 将生成的和弦索引转换为和弦名称。 返回生成的和弦序列。
* 引入包： time,pickle,utils.filePath,
* 调用： context\_to\_index,choice,
* 内部依赖描述：
  + context\_to\_index: 该函数用于将情绪标签和风格标签转换为上下文索引。它首先对情绪标签进行排序并转换为元组，然后与风格标签一起组成上下文元组。最后，通过上下文元组在self.context\_index字典中查找对应的索引，如果存在则返回索引，否则返回-1。

### **init** (service/ChordVectorAnnoy.py)

* 行号位置：9-25
* 重要性评分：6.70
* 功能描述： 初始化一个用于存储向量索引和ID映射的类，支持加载已有的索引和映射文件，或者从头开始创建新的索引和映射。
* 实现流程： 初始化类的属性，包括向量维度、索引路径和ID映射路径。 创建一个AnnoyIndex对象，用于存储向量索引，使用汉明距离作为相似度度量。 检查索引文件和ID映射文件是否存在。 如果文件存在，加载索引和ID映射，并设置下一个可用的ID为当前最大ID加一，然后卸载索引以便后续添加新数据。 如果文件不存在，初始化下一个可用的ID为0，准备从头开始创建索引和映射。
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： AnnoyIndex,exists,keys,unbuild,
* 内部依赖描述：

### updateButtonColor (VirtualKeyboard.py)

* 行号位置：413-439
* 重要性评分：6.70
* 功能描述： 该函数用于根据键盘按键的状态更新按钮的颜色和样式。当按键被按下时，根据按键的类型（白键或黑键）改变按钮的样式，使其显示为灰色边框和圆角，背景颜色为#878787或#787878。当按键未被按下时，恢复按钮的默认样式。
* 实现流程： 遍历所有按键，检查每个按键是否被按下。 如果按键被按下且为白键，则调用changeWhiteSheet函数，传入True参数，改变按钮样式。 如果按键被按下且为黑键，则调用changeBlackSheet函数，传入True参数，改变按钮样式。 如果按键未被按下且为白键，则调用changeWhiteSheet函数，传入False参数，恢复按钮样式。 如果按键未被按下且为黑键，则调用changeBlackSheet函数，传入False参数，恢复按钮样式。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： changeWhiteSheet,changeBlackSheet,
* 内部依赖描述：
  + changeWhiteSheet: 根据传入的isGray参数，改变按钮的样式。如果isGray为True，则设置按钮的边框为灰色，圆角为3px，并且背景颜色为#878787。如果isGray为False，则设置按钮的边框为灰色，圆角为3px，并且背景颜色为self.white\_color。
  + changeBlackSheet: 根据传入的isGray参数，改变按钮的样式。如果isGray为True，则按钮的背景颜色为#787878；如果isGray为False，则按钮的背景颜色为self.black\_color。

### **init** (service/MidiInput.py)

* 行号位置：25-35
* 重要性评分：6.10
* 功能描述： 该类用于初始化一个支持MIDI输入和键盘映射的系统，主要用于和弦序列预测，并支持情绪和风格标签作为上下文。
* 实现流程： 初始化父类 创建SoundNoise对象 初始化一个集合用于跟踪按下的键 设置是否使用键盘映射 设置运行状态为True 初始化Pygame和Pygame MIDI模块 调用内部函数初始化MIDI输入设备或启用键盘映射模式
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： **init**,SoundNoise,init,\_initialize\_midi\_input,
* 内部依赖描述：
  + **init**: 该类用于初始化一个支持MIDI输入和键盘映射的系统，主要用于和弦序列预测，并支持情绪和风格标签作为上下文。它设置了一个窗口标题和大小，并初始化了一些变量来控制图像更新。类中还设置了标签来显示图像，并通过布局管理器将标签添加到窗口中。
  + \_initialize\_midi\_input: 该函数用于初始化MIDI输入设备或启用键盘映射。它首先打印所有可用的MIDI设备信息，然后让用户选择一个设备。如果用户选择了一个设备，则初始化该设备；如果用户没有选择设备，则启用键盘映射模式。

### search\_chord (service/ChordVectorFaiss.py)

* 行号位置：95-115
* 重要性评分：6.10
* 功能描述： 该函数根据给定的 MIDI 音符序列搜索最相似的和弦。
* 实现流程： 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。 将转换后的向量转换为 numpy 数组，并设置数据类型为 float32。 使用索引搜索与输入向量最相似的和弦，返回相似度和索引。 根据索引从映射表中获取和弦信息，包括和弦名称、按压字符串和相似度。 将获取的和弦信息格式化为字典，并添加到结果列表中。 返回包含最相似和弦信息的结果列表。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： midi\_notes\_to\_vector,array,search,get,
* 内部依赖描述：
  + midi\_notes\_to\_vector: 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。

### \_on\_key\_press (service/MidiInput.py)

* 行号位置：79-95
* 重要性评分：5.70
* 功能描述： 该函数处理键盘按下事件，当焦点不在 QLineEdit 上时，根据按键字符执行相应的操作。如果按键是反引号（` 或 ~），则切换键盘监听开关。如果监听启用且按键在音符映射中，则调用\_handle\_note\_press函数处理音符按下事件。
* 实现流程： 获取当前焦点的控件。 检查焦点是否在 QLineEdit 上，如果是，则不触发 MIDI 音符映射。 检查按键字符是否为反引号（` 或 ~），如果是，则切换键盘监听开关，并记录日志。 如果监听启用且按键在音符映射中，则将按键字符转换为大写，并检查该音符是否在活动键集合中。 如果音符不在活动键集合中，则将其添加到集合中，并触发音符按下事件。 忽略特殊键（如 Shift 等）的异常情况。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： focusWidget,info,upper,\_handle\_note\_press,
* 内部依赖描述：
  + \_handle\_note\_press: 处理按下音符，如果音符不在活动键集合中，则将其添加到集合中，并触发音符按下事件。

### predict\_chord (service/numpyMarkov.py)

* 行号位置：73-95
* 重要性评分：5.30
* 功能描述： 该函数根据当前和弦列表预测下一个和弦，并过滤出概率大于阈值的和弦。它首先检查当前和弦的数量是否少于马尔科夫链的阶数，然后将当前和弦状态转换为索引，获取状态转移概率，并过滤和排序概率大于阈值的和弦，最后返回按概率降序排列的和弦及其概率。
* 实现流程： 检查当前和弦的数量是否少于马尔科夫链的阶数，如果少于则抛出异常。 将当前和弦状态转换为对应的索引。 获取当前和弦状态的转移概率。 过滤出概率大于阈值的和弦。 按概率降序排序过滤后的和弦。 规范化排序后的和弦概率。 返回按概率降序排列的和弦及其概率。
* 调用： ValueError,state\_to\_index,keys,
* 内部依赖描述：
  + state\_to\_index: 将和弦状态转换为对应的索引，用于状态表示和状态转移的计算。

### predict\_chord (service/markov.py)

* 行号位置：32-54
* 重要性评分：5.30
* 功能描述： 该函数用于预测音乐中的下一个和弦。它通过分析当前和弦序列来确定下一个和弦，并计算该和弦的概率。
* 实现流程： 获取当前状态：根据当前和弦序列和马尔科夫链的长度，确定当前状态。 匹配马尔科夫链：检查当前状态是否存在于马尔科夫链中。 选择下一个和弦：如果当前状态存在，从马尔科夫链中选择下一个和弦；否则，从默认的和弦列表中随机选择。 计算概率：根据选择的和弦在马尔科夫链中的出现次数，计算其概率。 更新状态：将当前状态和选择的和弦更新为新的状态，并获取新的马尔科夫链状态。 返回结果：返回选择的和弦及其概率。
* 引入包： random,
* 调用： choice,get,
* 内部依赖描述：

### \_initialize\_midi\_input (service/MidiInput.py)

* 行号位置：37-47
* 重要性评分：5.10
* 功能描述： 该函数用于初始化MIDI输入设备或启用键盘映射。它首先打印所有可用的MIDI设备信息，然后让用户选择一个设备。如果用户选择了一个设备，则初始化该设备；如果用户没有选择设备，则启用键盘映射模式。
* 实现流程： 打印所有可用的MIDI设备信息。 让用户手动选择一个MIDI设备。 如果用户选择了一个设备，则初始化该设备。 如果用户没有选择设备，则启用键盘映射模式。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： \_print\_midi\_devices,\_choose\_device,\_enable\_keyboard\_mapping,\_initialize\_midi\_device,
* 内部依赖描述：
  + \_print\_midi\_devices: 该函数用于打印所有可用的MIDI设备的信息。
  + \_choose\_device: 该函数用于手动选择MIDI设备。用户可以通过输入设备ID来指定设备，如果不输入则跳过使用键盘映射。
  + \_enable\_keyboard\_mapping: 启用键盘映射模式，记录日志并启动键盘监听器。
  + \_initialize\_midi\_device: 该函数用于初始化MIDI设备，通过传入设备ID来创建一个MIDI输入对象，并设置是否使用键盘映射。

### delete\_chord (service/ChordVectorAnnoy.py)

* 行号位置：72-81
* 重要性评分：5.00
* 功能描述： 该函数用于从索引中删除指定的和弦，并更新相关的映射和索引。
* 实现流程： 根据和弦名称查找所有相关的索引ID。 遍历这些索引ID，逐个从索引中移除对应的项，并从映射字典中删除这些ID。 在所有项被移除后，重新构建索引以确保索引和映射的一致性。
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： items,unbuild,remove\_item,build\_index,
* 内部依赖描述：
  + build\_index: 该函数用于构建索引并保存索引和映射文件。

### generate\_chord\_sequence (service/ChordHMM.py)

* 行号位置：193-212
* 重要性评分：5.00
* 功能描述： 该函数使用训练好的隐马尔可夫模型（HMM）生成指定长度的和弦序列。
* 实现流程： 初始化状态和观察结果列表。 随机选择一个初始状态，根据初始状态概率分布。 循环生成指定长度的和弦序列： 根据当前状态，从状态转移概率矩阵中选择下一个状态。 根据当前状态，从观察概率矩阵中选择一个和弦。 将选择的状态和和弦分别添加到状态和观察结果列表中。 将观察结果列表中的索引和弦转换为实际的和弦名称。 返回生成的和弦序列。
* 引入包： pickle,utils.filePath,
* 调用： choice,
* 内部依赖描述：

### predict\_next\_chord (service/ChordHMM.py)

* 行号位置：173-191
* 重要性评分：4.90
* 功能描述： 该函数根据当前和弦序列预测下一个和弦。它首先将和弦序列转换为对应的索引序列，然后使用Viterbi算法找到最可能的隐藏状态序列，最后根据该状态序列预测下一个和弦。
* 实现流程： 将当前和弦序列转换为对应的索引序列。 检查索引序列中是否包含未知和弦，如果包含则返回None。 使用Viterbi算法找到最可能的隐藏状态序列。 根据最可能的隐藏状态序列预测下一个和弦的概率分布。 选择概率最高的和弦作为预测结果并返回。
* 引入包： pickle,utils.filePath,
* 调用： chords\_to\_indices,viterbi,argmax,
* 内部依赖描述：
  + chords\_to\_indices: 将和弦序列转换为对应的索引序列。
  + viterbi: Viterbi算法用于在给定观察序列的情况下，找到最可能的隐藏状态序列。

### **init** (service/ChordHMM.py)

* 行号位置：9-27
* 重要性评分：4.90
* 功能描述： 初始化一个用于和弦序列预测的HMM模型。该模型可以处理和弦序列，并为每个和弦分配一个唯一的索引。如果提供了和弦序列，则会初始化模型的参数；如果没有提供，则模型将为空，等待后续加载。
* 实现流程： 检查是否提供了和弦序列。 如果提供了和弦序列，初始化隐藏状态数。 为每个和弦分配一个唯一的索引，并创建和弦到索引的映射字典。 计算不同和弦的数量。 创建索引到和弦的映射字典。 初始化HMM模型的参数。 如果未提供和弦序列，初始化模型为空，等待后续加载。
* 引入包： pickle,utils.filePath,
* 调用： index\_chords,items,initialize\_parameters,
* 内部依赖描述：
  + index\_chords: 该函数用于为和弦序列中的每个唯一和弦分配一个唯一的索引。
  + initialize\_parameters: 该函数用于随机初始化隐马尔可夫模型（HMM）的参数，包括状态转移矩阵A、发射矩阵B和初始状态概率pi。

### reShaderLists (VirtualKeyboard.py)

* 行号位置：337-343
* 重要性评分：4.70
* 功能描述： 该函数用于清空列表小部件和堆叠小部件中的所有内容，并重新加载新的着色器列表。
* 实现流程： 清空列表小部件中的所有项。 遍历堆叠小部件中的所有小部件，并逐个移除。 调用内部函数 shaderLists，传入 vbox、列表小部件和堆叠小部件作为参数，重新加载着色器列表。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： clear,widget,removeWidget,shaderLists,
* 内部依赖描述：
  + clear: 该函数用于清空对象的数组属性。
  + shaderLists: 该函数用于在GUI中创建一个包含标注库和历史记录的列表视图，并将对应的文本内容显示在堆叠窗口中。它会读取指定目录下的文件，并将文件名和内容分别添加到列表和堆叠窗口中，同时实现列表和堆叠窗口之间的交互。

### build\_transition\_matrix (service/numpyMarkov.py)

* 行号位置：31-55
* 重要性评分：4.50
* 功能描述： 该函数根据输入的和弦序列构建一个概率转移矩阵，用于表示和弦之间的转移概率。
* 实现流程： 遍历输入的和弦序列，对于每个和弦序列，计算其马尔科夫链阶数的转移概率。 将当前和弦状态转换为对应的索引，用于状态表示和状态转移的计算。 获取下一个和弦的索引，处理未知和弦的情况。 更新转移矩阵中当前状态到下一个状态的计数。 对转移矩阵的每一行进行归一化处理，将计数归一化为概率。 最终得到一个表示和弦之间转移概率的矩阵。
* 调用： state\_to\_index,get,
* 内部依赖描述：
  + state\_to\_index: 将和弦状态转换为对应的索引，用于状态表示和状态转移的计算。

### find\_matching\_scales (service/MatchingScales.py)

* 行号位置：41-64
* 重要性评分：4.40
* 功能描述： 该函数用于找到与输入音符集匹配的音阶。它首先将根音转换为 MIDI 音高类编号，然后合并所有和弦的音符并去重。接着，它遍历预定义的音阶，将每个音阶转调到根音，并计算与输入音符集的匹配度。最后，它按匹配度排序并返回匹配的音阶列表。
* 实现流程： 将根音转换为 MIDI 音高类编号。 合并所有和弦的音符并去重，得到输入音符集的音高类集合。 遍历预定义的音阶，将每个音阶转调到根音。 计算每个转调音阶与输入音符集的匹配度。 将匹配度大于等于阈值的音阶及其匹配度添加到匹配音阶列表中。 按匹配度对匹配音阶列表进行排序。 返回排序后的匹配音阶列表。
* 调用： items,sort,
* 内部依赖描述：

### initialize\_parameters (service/ChordHMMV2.py)

* 行号位置：68-80
* 重要性评分：4.30
* 功能描述： 该函数用于随机初始化隐马尔可夫模型（HMM）的参数，包括状态转移矩阵A、发射矩阵B和初始状态概率pi。发射矩阵B是三维的，考虑了上下文依赖。
* 实现流程： 生成一个随机的状态转移矩阵A，其维度为(n\_states, n\_states)。 对A的每一行进行归一化，确保每一行的和为1。 生成一个随机的发射矩阵B，其维度为(n\_states, n\_chords, n\_contexts)。 对B的每一行进行归一化，确保每一行的和为1。 生成一个随机的初始状态概率向量pi，其维度为(n\_states,)。 对pi进行归一化，确保其和为1。
* 引入包： time,pickle,utils.filePath,
* 调用： rand,
* 内部依赖描述：

### import\_data (service/ChordVectorFaiss.py)

* 行号位置：154-165
* 重要性评分：4.20
* 功能描述： 该函数用于从缓存文件中导入和弦数据，并将其添加到索引中，然后保存索引和映射。
* 实现流程： 打开缓存文件进行读取。 逐行读取文件内容，去除行首尾的空白字符。 跳过空行。 将每一行按 ‘::’ 分割成和弦名称和按弦字符串列表。 将和弦名称和每个按弦字符串传递给 add\_chord 函数，将其添加到索引中并转换为 MIDI 音符序列的 One-Hot 编码向量。 遍历所有处理过的和弦和按弦字符串后，调用 save\_index 函数保存索引和映射。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： strip,add\_chord,save\_index,
* 内部依赖描述：
  + add\_chord: 该函数用于将和弦名称和按弦字符串添加到索引中，并将其转换为 MIDI 音符序列的 One-Hot 编码向量。
  + save\_index: 该函数用于保存索引和映射。它将索引写入指定路径，并将映射对象序列化后保存到另一个指定路径。

### data\_cut (service/ChordVectorAnnoy.py)

* 行号位置：92-103
* 重要性评分：4.20
* 功能描述： 该函数用于从缓存文件中读取和弦数据，并将和弦名称和按弦字符串添加到索引中，最终构建索引。
* 实现流程： 打开缓存文件进行读取。 逐行读取文件内容，去除行首尾的空白字符。 跳过空行。 将每一行按 ‘::’ 分割成和弦名称和按弦字符串列表。 遍历按弦字符串列表，调用 add\_chord 函数将和弦名称和按弦字符串添加到索引中，并转换为 MIDI 音符序列的 One-Hot 编码向量。 所有数据处理完毕后，调用 build\_index 函数构建索引。
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： strip,add\_chord,build\_index,
* 内部依赖描述：
  + add\_chord: 该函数用于将和弦名称和按弦字符串添加到索引中，并将其转换为 MIDI 音符序列的 One-Hot 编码向量。
  + build\_index: 该函数用于构建索引并保存索引和映射文件。

### initialize\_parameters (service/ChordHMM.py)

* 行号位置：40-51
* 重要性评分：4.20
* 功能描述： 该函数用于随机初始化隐马尔可夫模型（HMM）的参数，包括状态转移矩阵A、发射矩阵B和初始状态概率pi。
* 实现流程： 生成一个随机的n\_states x n\_states的矩阵A，并对其进行行归一化，确保每行的和为1。 生成一个随机的n\_states x n\_chords的矩阵B，并对其进行行归一化，确保每行的和为1。 生成一个随机的n\_states的向量pi，并对其进行归一化，确保其和为1。
* 引入包： pickle,utils.filePath,
* 调用： rand,
* 内部依赖描述：

### \_on\_key\_release (service/MidiInput.py)

* 行号位置：97-107
* 重要性评分：4.10
* 功能描述： 该函数用于处理键盘释放事件，当焦点不在文本输入框时，根据释放的键字符查找对应的音符，并调用\_handle\_note\_release函数处理音符释放事件。
* 实现流程： 获取当前焦点的控件。 检查焦点控件是否为QLineEdit文本输入框，如果是，则不触发音符释放事件。 尝试获取释放的键字符，并将其转换为大写。 检查键字符是否在预定义的键映射中。 如果键字符在映射中，调用\_handle\_note\_release函数处理音符释放事件，将音符从活动键列表中移除并触发键释放事件。 如果键字符不在映射中，捕获AttributeError异常并忽略该事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： focusWidget,upper,\_handle\_note\_release,
* 内部依赖描述：
  + \_handle\_note\_release: 处理释放音符，当音符在活动键列表中时，将其移除并触发键释放事件。

### **init** (service/numpyMarkov.py)

* 行号位置：6-16
* 重要性评分：4.10
* 功能描述： 该类用于初始化一个马尔科夫链预测器，用于预测和弦序列。它通过为和弦分配唯一索引并构建转移概率矩阵来实现这一功能。
* 实现流程： 初始化预测器，设置马尔科夫链的阶数。 构建和弦索引字典，将每个和弦映射到一个唯一的整数索引，并为未知和弦分配一个独特的索引。 计算唯一和弦的数量。 初始化概率转移矩阵，大小为 (n\_chords^order, n\_chords)。 根据输入的和弦序列构建转移概率矩阵，表示和弦之间的转移概率。
* 调用： index\_chords,zeros,build\_transition\_matrix,
* 内部依赖描述：
  + index\_chords: 该函数用于为和弦序列中的每个唯一和弦分配一个唯一的索引。
  + build\_transition\_matrix: 该函数根据输入的和弦序列构建一个概率转移矩阵，用于表示和弦之间的转移概率。

### \_process\_midi\_events (service/MidiInput.py)

* 行号位置：142-151
* 重要性评分：4.00
* 功能描述： 该函数用于处理MIDI事件，包括音符按下和释放事件。它会解析每个MIDI事件的状态、音符和力度，并根据状态调用相应的内部函数来处理音符的按下和释放。
* 实现流程： 遍历传入的MIDI事件列表。 解析每个事件的状态、音符和力度。 根据状态判断事件是音符按下还是释放。 如果事件是音符按下且力度大于0，调用\_handle\_note\_press函数处理音符按下事件。 如果事件是音符释放或音符按下但力度为0，调用\_handle\_note\_release函数处理音符释放事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： info,\_handle\_note\_press,\_handle\_note\_release,
* 内部依赖描述：
  + \_handle\_note\_press: 处理按下音符，如果音符不在活动键集合中，则将其添加到集合中，并触发音符按下事件。
  + \_handle\_note\_release: 处理释放音符，当音符在活动键列表中时，将其移除并触发键释放事件。

### virtual\_key\_released (VirtualKeyboard.py)

* 行号位置：555-564
* 重要性评分：4.00
* 功能描述： 处理虚拟钢琴键盘释放事件，更新按键状态并触发相关操作。
* 实现流程： 检查按键索引是否在有效范围内（0到87）。 如果有效，将对应按键的状态更新为未按下。 调用更新方法以反映按键状态的变化。 触发按键释放事件。 更新按钮的颜色以反映当前按键状态。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： update,pressingEvent,updateButtonColor,
* 内部依赖描述：
  + pressingEvent: 该函数用于检测当前按下的琴键，并根据按下的琴键数量和音符组合来确定当前的和弦。如果按下的琴键数量在3到5之间，它会调用音乐工具的检测函数来识别和弦，并更新UI显示当前和弦。如果按下的琴键数量为0，它会检查是否有之前的和弦记录，并根据记录的和弦与当前和弦是否相同来决定是否更新队列。如果和弦不同，它会根据队列的长度决定是添加新和弦还是替换旧和弦，并启动异步线程将和弦和对应的音符写入本地文件。
  + updateButtonColor: 该函数用于根据键盘按键的状态更新按钮的颜色和样式。当按键被按下时，根据按键的类型（白键或黑键）改变按钮的样式，使其显示为灰色边框和圆角，背景颜色为#878787或#787878。当按键未被按下时，恢复按钮的默认样式。

### virtual\_key\_pressed (VirtualKeyboard.py)

* 行号位置：544-553
* 重要性评分：4.00
* 功能描述： 处理虚拟钢琴键盘按下事件，更新按键状态并触发相关操作。
* 实现流程： 检查按键索引是否在有效范围内（0到87）。 如果有效，更新对应按键的状态为按下（True）。 调用update方法更新界面显示。 触发pressingEvent事件，可能用于处理按键按下时的逻辑。 调用updateButtonColor方法更新按键颜色，可能用于视觉反馈。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： update,pressingEvent,updateButtonColor,
* 内部依赖描述：
  + pressingEvent: 该函数用于检测当前按下的琴键，并根据按下的琴键数量和音符组合来确定当前的和弦。如果按下的琴键数量在3到5之间，它会调用音乐工具的检测函数来识别和弦，并更新UI显示当前和弦。如果按下的琴键数量为0，它会检查是否有之前的和弦记录，并根据记录的和弦与当前和弦是否相同来决定是否更新队列。如果和弦不同，它会根据队列的长度决定是添加新和弦还是替换旧和弦，并启动异步线程将和弦和对应的音符写入本地文件。
  + updateButtonColor: 该函数用于根据键盘按键的状态更新按钮的颜色和样式。当按键被按下时，根据按键的类型（白键或黑键）改变按钮的样式，使其显示为灰色边框和圆角，背景颜色为#878787或#787878。当按键未被按下时，恢复按钮的默认样式。

### save\_model (service/ChordHMMV2.py)

* 行号位置：315-333
* 重要性评分：3.90
* 功能描述： 该函数用于将模型参数保存到指定的文件路径中。
* 实现流程： 定义一个包含模型所有参数的字典model\_data。 使用pickle模块将model\_data字典以二进制格式写入到指定的文件路径中。 打印保存成功的消息，显示保存的文件路径。
* 引入包： time,pickle,utils.filePath,
* 调用： filePath,dump,
* 内部依赖描述：
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。

### backward (service/ChordHMMV2.py)

* 行号位置：122-139
* 重要性评分：3.80
* 功能描述： 后向算法用于计算贝塔值，贝塔值在隐马尔可夫模型中用于计算从当前时间步到序列末尾的概率。
* 实现流程： 确定观察序列O的长度T和状态数N。 初始化贝塔矩阵beta，其中beta[T-1]设为全1，因为从最后一个时间步到序列末尾的概率为1。 从倒数第二个时间步开始，向前遍历到第一个时间步。 对于每个时间步t和每个状态i，计算贝塔值beta[t, i]，它等于从时间步t+1到序列末尾的所有可能状态j的贝塔值beta[t+1, j]乘以状态转移概率A[i, j]和发射概率B[j, O[t+1], context\_index]的乘积之和。 返回计算得到的贝塔矩阵beta。
* 引入包： time,pickle,utils.filePath,
* 调用： zeros,ones,
* 内部依赖描述：

### test\_exception\_for\_insufficient\_chords (utils/tests.py)

* 行号位置：51-58
* 重要性评分：3.80
* 功能描述： 该函数测试了在和弦序列长度不足模型阶数时，ChordPredictor类的predict\_chord方法是否会抛出异常。
* 实现流程： 定义了一个包含两个和弦序列的列表sequences。 创建了一个ChordPredictor对象predictor，使用sequences作为训练数据，并设置模型的阶数为2。 尝试调用predict\_chord方法，传入一个只包含一个和弦的列表[‘C’]。 由于当前和弦数量少于模型的阶数，predict\_chord方法应该抛出ValueError异常。 捕获异常并打印异常信息，验证异常情况测试通过。
* 引入包： unittest,service.numpyMarkov,
* 调用： ChordPredictor,predict\_chord,AssertionError,
* 内部依赖描述：
  + predict\_chord: 该函数用于预测音乐中的下一个和弦。它通过分析当前和弦序列来确定下一个和弦，并计算该和弦的概率。

### test\_extended\_sequences (utils/tests.py)

* 行号位置：64-81
* 重要性评分：3.80
* 功能描述： 该函数用于测试ChordPredictor类的predict\_chord方法，通过给定的扩充和弦序列和当前和弦，预测下一个和弦。
* 实现流程： 定义一个包含多个扩充和弦序列的列表。 创建一个ChordPredictor对象，传入扩充和弦序列列表和order参数。 设置当前和弦为[‘D’]。 调用predict\_chord方法，传入当前和弦，获取预测结果。 断言预测结果不为空，如果为空则测试失败。 如果预测结果非空，则测试通过，并打印预测结果。
* 引入包： unittest,service.numpyMarkov,
* 调用： ChordPredictor,predict\_chord,
* 内部依赖描述：
  + predict\_chord: 该函数用于预测音乐中的下一个和弦。它通过分析当前和弦序列来确定下一个和弦，并计算该和弦的概率。

### update\_chord (service/ChordVectorAnnoy.py)

* 行号位置：83-89
* 重要性评分：3.70
* 功能描述： 该函数用于更新指定和弦的 MIDI 音符序列。它首先删除指定和弦的旧数据，然后添加新的和弦数据，并重新构建索引。
* 实现流程： 删除指定和弦的旧数据 添加新的和弦数据，包括和弦名称和按弦字符串 将新的和弦数据转换为 MIDI 音符序列的 One-Hot 编码向量 重新构建索引以包含新的和弦数据
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： delete\_chord,add\_chord,build\_index,
* 内部依赖描述：
  + delete\_chord: 该函数用于从索引中删除指定的和弦，并更新相关的映射和索引。
  + add\_chord: 该函数用于将和弦名称和按弦字符串添加到索引中，并将其转换为 MIDI 音符序列的 One-Hot 编码向量。
  + build\_index: 该函数用于构建索引并保存索引和映射文件。

### save\_model (service/ChordHMM.py)

* 行号位置：237-253
* 重要性评分：3.70
* 功能描述： 该函数用于将模型参数保存到指定的文件路径。
* 实现流程： 定义一个包含模型参数的字典model\_data。 使用pickle库将model\_data字典以二进制格式写入到指定的文件路径。 打印保存成功的消息，包含文件路径。
* 引入包： pickle,utils.filePath,
* 调用： filePath,dump,
* 内部依赖描述：
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。

### update\_chord (service/ChordVectorFaiss.py)

* 行号位置：145-151
* 重要性评分：3.70
* 功能描述： 该函数用于更新指定和弦的 MIDI 音符序列。它首先删除指定和弦，然后添加新的和弦及其按弦字符串，最后保存更新后的索引和映射。
* 实现流程： 删除指定和弦 添加新的和弦及其按弦字符串 将新的和弦转换为 MIDI 音符序列的 One-Hot 编码向量 保存更新后的索引和映射
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： delete\_chord,add\_chord,save\_index,
* 内部依赖描述：
  + delete\_chord: 该函数用于从索引中删除指定的和弦，并更新相关的映射和索引。
  + add\_chord: 该函数用于将和弦名称和按弦字符串添加到索引中，并将其转换为 MIDI 音符序列的 One-Hot 编码向量。
  + save\_index: 该函数用于保存索引和映射。它将索引写入指定路径，并将映射对象序列化后保存到另一个指定路径。

### search\_chord (service/ChordVectorAnnoy.py)

* 行号位置：55-70
* 重要性评分：3.60
* 功能描述： 该函数根据给定的 MIDI 音符序列搜索最相似的和弦，并返回这些和弦的名称、按压方式和与输入序列的距离。
* 实现流程： 将输入的 MIDI 音符序列转换为 One-Hot 编码的向量表示。 使用向量在索引中查找最相似的和弦ID及其距离。 根据找到的ID，从映射中获取和弦的名称和按压方式。 将结果以包含和弦名称、按压方式和距离的字典列表形式返回。
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： midi\_notes\_to\_vector,get\_nns\_by\_vector,
* 内部依赖描述：
  + midi\_notes\_to\_vector: 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。

### \_emit\_key\_release (service/MidiInput.py)

* 行号位置：128-133
* 重要性评分：3.60
* 功能描述： 该函数用于触发按键释放事件，将MIDI音符转换为虚拟键值，并通过信号发出按键释放事件，同时记录日志并调用内部函数处理MIDI输入事件。
* 实现流程： 计算虚拟键值：将输入的MIDI音符减去36，得到虚拟键值。 触发按键释放事件：通过信号v\_key\_released发出按键释放事件，传递虚拟键值。 记录日志：使用logging.info记录按键释放的日志信息，包含MIDI音符值。 处理MIDI输入事件：调用内部函数on\_midi\_input，传递MIDI音符、音量（0）和按键状态（False），以决定是否播放音符或关闭音符。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： emit,info,on\_midi\_input,
* 内部依赖描述：
  + on\_midi\_input: 该函数用于处理MIDI输入事件，根据输入的音符和音量信息，决定是播放音符还是关闭音符。

### \_emit\_key\_press (service/MidiInput.py)

* 行号位置：121-126
* 重要性评分：3.60
* 功能描述： 该函数用于触发按键按下事件，将MIDI音符转换为虚拟键码，并通过信号发出按键按下事件。同时，记录按键按下日志，并调用内部函数处理MIDI输入事件。
* 实现流程： 计算虚拟键码：将输入的MIDI音符减去36，得到虚拟键码。 触发按键按下事件：通过信号发出虚拟键码，通知其他部分按键已按下。 记录日志：使用logging记录按键按下事件的日志。 处理MIDI输入事件：调用内部函数on\_midi\_input，根据音符和音量信息决定是播放音符还是关闭音符。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： emit,info,on\_midi\_input,
* 内部依赖描述：
  + on\_midi\_input: 该函数用于处理MIDI输入事件，根据输入的音符和音量信息，决定是播放音符还是关闭音符。

### run (service/MidiInput.py)

* 行号位置：135-140
* 重要性评分：3.60
* 功能描述： 该函数用于持续监听MIDI输入事件，并在检测到事件时进行处理。
* 实现流程： 函数首先进入一个无限循环，直到self.running为False。 在循环中，检查是否启用了键盘映射（self.use\_keyboard\_mapping）。 如果未启用键盘映射且MIDI输入有事件（self.midi\_input.poll()返回True），则读取最多10个MIDI事件（self.midi\_input.read(10)）。 读取到的事件会被传递给私有方法self.\_process\_midi\_events进行处理。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： poll,read,\_process\_midi\_events,
* 内部依赖描述：
  + \_process\_midi\_events: 该函数用于处理MIDI事件，包括音符按下和释放事件。它会解析每个MIDI事件的状态、音符和力度，并根据状态调用相应的内部函数来处理音符的按下和释放。

### backward (service/ChordHMM.py)

* 行号位置：79-94
* 重要性评分：3.60
* 功能描述： 后向算法用于计算贝塔值，贝塔值在隐马尔可夫模型中用于计算给定观察序列下每个状态在每个时间步的概率。
* 实现流程： 确定观察序列O的长度T和状态转移矩阵A、发射矩阵B的维度N。 初始化贝塔矩阵beta，其形状为(T, N)，并将最后一行设为全1，因为最后一个时间步的贝塔值为1。 从倒数第二个时间步开始，向前遍历到第一个时间步。 对于每个时间步t和每个状态i，计算贝塔值beta[t, i]，它是所有可能的下一个状态j的贝塔值beta[t + 1, j]、状态转移概率A[i, j]和发射概率B[j, O[t + 1]]的乘积之和。 返回计算得到的贝塔矩阵beta。
* 引入包： pickle,utils.filePath,
* 调用： zeros,ones,
* 内部依赖描述：

### build\_index (service/ChordVectorAnnoy.py)

* 行号位置：49-53
* 重要性评分：3.50
* 功能描述： 该函数用于构建索引并保存索引和映射文件。
* 实现流程： 调用self.index.build(n\_trees)方法构建索引，其中n\_trees参数指定树的数量。 调用self.index.save(self.index\_path)方法将构建好的索引保存到指定路径。 使用pickle.dump将self.id\_mapping对象以二进制模式写入到self.mapping\_path指定的文件中。
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： build,save,dump,
* 内部依赖描述：

### onClear (VirtualKeyboard.py)

* 行号位置：395-398
* 重要性评分：3.40
* 功能描述： 该函数用于清空队列并重置所有按钮的文本。
* 实现流程： 调用self.QUEUE.clear()方法清空队列。 使用for循环遍历从0到self.MAX\_QUEUE-1的索引。 在每次循环中，使用self.findChild(QPushButton, ‘ch’ + str(i))方法找到对应的QPushButton对象。 调用找到的QPushButton对象的setText(’’)方法，将按钮的文本重置为空字符串。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： clear,findChild,setText,
* 内部依赖描述：
  + clear: 该函数用于清空对象的数组属性。

### add\_chord (service/ChordVectorAnnoy.py)

* 行号位置：37-47
* 重要性评分：3.10
* 功能描述： 该函数用于将和弦名称和按弦字符串添加到索引中，并将其转换为 MIDI 音符序列的 One-Hot 编码向量。
* 实现流程： 解析按弦字符串，提取根音符和音符序列。 将音符序列转换为 MIDI 音符列表。 将 MIDI 音符列表转换为 One-Hot 编码的向量表示。 将向量添加到索引中，并生成一个唯一的 ID。 将和弦名称、按弦字符串和 MIDI 音符列表与该 ID 关联存储。 递增下一个可用的 ID
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： midi\_notes\_to\_vector,add\_item,
* 内部依赖描述：
  + midi\_notes\_to\_vector: 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。

### forward (service/ChordHMMV2.py)

* 行号位置：101-120
* 重要性评分：3.00
* 功能描述： 该函数实现了前向算法，用于计算给定观察序列下的状态概率矩阵alpha。
* 实现流程： 初始化Alpha矩阵，其大小为T（观察序列长度）乘以N（状态数）。 根据初始状态概率pi和发射矩阵B，计算第一个时间步t=0时的状态概率alpha[0]。 从第二个时间步t=1开始，遍历整个观察序列，对于每个时间步t和每个状态j，计算状态概率alpha[t, j]，该概率由发射概率b和前一个时间步所有状态的概率乘以状态转移概率A得到。 最终返回计算得到的Alpha矩阵。
* 引入包： time,pickle,utils.filePath,
* 调用： zeros,
* 内部依赖描述：

### test\_threshold\_filtering (utils/tests.py)

* 行号位置：38-45
* 重要性评分：2.80
* 功能描述： 该函数用于测试基于阈值的和弦预测过滤功能。它通过设置一个较高的阈值来过滤预测结果，确保只有概率高于该阈值的和弦被保留。
* 实现流程： 定义一个包含多个和弦序列的列表。 使用ChordPredictor类初始化预测器，并传入和弦序列列表和阶数。 设置一个较高的阈值，以确保过滤效果明显。 调用predict\_chord方法，传入起始和弦和阈值，获取预测结果。 遍历预测结果，检查每个和弦的概率是否高于阈值，如果低于则断言失败。 如果所有预测结果的概率都高于阈值，则通过测试并打印预测结果。
* 引入包： unittest,service.numpyMarkov,
* 调用： ChordPredictor,predict\_chord,
* 内部依赖描述：
  + predict\_chord: 该函数用于预测音乐中的下一个和弦。它通过分析当前和弦序列来确定下一个和弦，并计算该和弦的概率。

### test\_different\_orders (utils/tests.py)

* 行号位置：25-32
* 重要性评分：2.80
* 功能描述： 该函数测试了不同阶数的和弦预测功能。它使用两个和弦序列，每个序列对应一个阶数，并通过ChordPredictor类进行预测。预测结果将被验证以确保预测成功。
* 实现流程： 定义两个和弦序列，每个序列对应一个阶数。 遍历不同的阶数（1和2）。 对于每个阶数，创建一个ChordPredictor实例，并传入相应的和弦序列和阶数。 根据当前和弦序列（’G’或[‘G’, ‘Am’]）进行预测。 验证预测结果是否为空，如果为空则抛出异常。 打印预测测试通过的信息，并输出最终的预测结果。
* 引入包： unittest,service.numpyMarkov,
* 调用： ChordPredictor,predict\_chord,
* 内部依赖描述：
  + predict\_chord: 该函数用于预测音乐中的下一个和弦。它通过分析当前和弦序列来确定下一个和弦，并计算该和弦的概率。

### load\_model (service/ChordHMMV2.py)

* 行号位置：335-351
* 重要性评分：2.70
* 功能描述： 该函数用于从指定路径加载模型参数，并将这些参数赋值给类的实例变量。
* 实现流程： 打开指定路径的模型文件。 使用pickle模块加载模型数据。 将加载的模型数据中的各个参数赋值给类的实例变量。 打印加载成功的消息。
* 引入包： time,pickle,utils.filePath,
* 调用： filePath,
* 内部依赖描述：
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。

### stop (service/MidiInput.py)

* 行号位置：153-159
* 重要性评分：2.70
* 功能描述： 停止MIDI输入并清理相关资源。
* 实现流程： 将self.running设置为False，表示停止运行。 如果self.use\_keyboard\_mapping为False，则关闭MIDI输入并退出pygame.midi。 退出pygame。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： quit,
* 内部依赖描述：

### \_initialize\_midi\_device (service/MidiInput.py)

* 行号位置：71-77
* 重要性评分：2.70
* 功能描述： 该函数用于初始化MIDI设备，通过传入设备ID来创建一个MIDI输入对象，并设置是否使用键盘映射。
* 实现流程： 尝试使用传入的设备ID初始化MIDI输入设备。 如果初始化成功，设置use\_keyboard\_mapping为False。 如果初始化失败，抛出一个包含错误信息的ValueError异常。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： Input,ValueError,
* 内部依赖描述：

### forward (service/ChordHMM.py)

* 行号位置：61-77
* 重要性评分：2.70
* 功能描述： 前向算法用于计算给定观察序列在隐马尔可夫模型中的概率，通过计算Alpha矩阵来实现。
* 实现流程： 初始化Alpha矩阵，其第一行由初始状态概率和发射矩阵相乘得到。 从第二行开始，遍历每个时间步t，计算每个状态j在时间t的概率，该概率由发射矩阵和前一时间步所有状态的概率与状态转移矩阵相乘得到。 最终返回Alpha矩阵，其中每一行代表一个时间步，每一列代表一个状态，值表示在该时间步处于该状态的概率。
* 引入包： pickle,utils.filePath,
* 调用： zeros,
* 内部依赖描述：

### save\_index (service/ChordVectorFaiss.py)

* 行号位置：87-93
* 重要性评分：2.70
* 功能描述： 该函数用于保存索引和映射。它将索引写入指定路径，并将映射对象序列化后保存到另一个指定路径。
* 实现流程： 使用faiss.write\_index方法将索引保存到self.index\_path指定的路径。 打开self.mapping\_path指定的路径，以二进制写模式打开。 使用pickle.dump方法将self.id\_mapping对象序列化并写入文件。 关闭文件以确保数据正确写入磁盘。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： write\_index,dump,
* 内部依赖描述：

### changeBlackSheet (VirtualKeyboard.py)

* 行号位置：514-520
* 重要性评分：2.70
* 功能描述： 根据传入的isGray参数，改变按钮的样式。如果isGray为True，则按钮的背景颜色为#787878；如果isGray为False，则按钮的背景颜色为self.black\_color。
* 实现流程： 检查isGray参数的值。 如果isGray为True，则设置按钮的样式，背景颜色为#787878。 如果isGray为False，则设置按钮的样式，背景颜色为self.black\_color。 在设置样式时，按钮的边框样式、圆角和左边距保持不变。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： setStyleSheet,
* 内部依赖描述：

### changeWhiteSheet (VirtualKeyboard.py)

* 行号位置：507-512
* 重要性评分：2.60
* 功能描述： 根据传入的isGray参数，改变按钮的样式。如果isGray为True，则设置按钮的边框为灰色，圆角为3px，并且背景颜色为#878787。如果isGray为False，则设置按钮的边框为灰色，圆角为3px，并且背景颜色为self.white\_color。
* 实现流程： 检查isGray参数的值。 如果isGray为True，则设置按钮的样式为：边框为1px灰色，圆角为3px，背景颜色为#878787。 如果isGray为False，则设置按钮的样式为：边框为1px灰色，圆角为3px，背景颜色为self.white\_color。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： setStyleSheet,
* 内部依赖描述：

### test\_basic\_prediction (utils/tests.py)

* 行号位置：14-19
* 重要性评分：2.60
* 功能描述： 该函数用于测试基本的和弦预测功能。它使用一个包含两个和弦序列的列表来训练一个和弦预测器，并使用该预测器对给定的和弦序列进行预测。
* 实现流程： 定义一个包含两个和弦序列的列表。 使用这些序列训练一个和弦预测器，设置预测器的阶数为1。 使用训练好的预测器对给定的和弦序列[‘G’]进行预测。 检查预测结果是否为空，如果不为空，则打印预测结果并标记测试通过。
* 引入包： unittest,service.numpyMarkov,
* 调用： ChordPredictor,predict\_chord,
* 内部依赖描述：
  + predict\_chord: 该函数用于预测音乐中的下一个和弦。它通过分析当前和弦序列来确定下一个和弦，并计算该和弦的概率。

### \_enable\_keyboard\_mapping (service/MidiInput.py)

* 行号位置：60-64
* 重要性评分：2.50
* 功能描述： 启用键盘映射模式，记录日志并启动键盘监听器。
* 实现流程： 记录日志，提示使用键盘映射模式。 设置内部状态，启用键盘映射模式。 启动键盘监听器，开始监听键盘事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： info,\_start\_keyboard\_listener,
* 内部依赖描述：
  + \_start\_keyboard\_listener: 该函数用于启动一个键盘事件监听器，监听键盘按键的按下和释放事件，并在事件发生时调用相应的处理函数。

### \_print\_midi\_devices (service/MidiInput.py)

* 行号位置：49-53
* 重要性评分：2.50
* 功能描述： 该函数用于打印所有可用的MIDI设备的信息。
* 实现流程： 使用pygame.midi.get\_count()获取可用的MIDI设备数量。 遍历每个设备，使用pygame.midi.get\_device\_info(i)获取设备信息。 打印每个设备的索引和信息。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： get\_count,get\_device\_info,
* 内部依赖描述：

### on\_midi\_input (service/soundNoise.py)

* 行号位置：45-49
* 重要性评分：2.50
* 功能描述： 该函数用于处理MIDI输入事件，根据输入的音符和音量信息，决定是播放音符还是关闭音符。
* 实现流程： 接收MIDI输入事件，包含音符（note）、音量（velocity）和是否为音符开启（is\_note\_on）。 检查is\_note\_on参数，如果为True，则调用play\_note\_on函数，传入note和velocity参数，播放指定音符。 如果is\_note\_on参数为False，则调用play\_note\_off函数，传入note参数，关闭指定音符。
* 引入包： platform,fluidsynth,utils.filePath,logging,
* 调用： play\_note\_on,play\_note\_off,
* 内部依赖描述：
  + play\_note\_on: 该函数用于在乐器上播放指定音符，音符的音高由参数note决定，音量由参数velocity决定，默认音量为127。
  + play\_note\_off: 关闭指定音符

### state\_to\_index (service/numpyMarkov.py)

* 行号位置：57-71
* 重要性评分：2.50
* 功能描述： 将和弦状态转换为对应的索引，用于状态表示和状态转移的计算。
* 实现流程： 初始化索引为0。 遍历当前和弦状态中的每个和弦。 对于每个和弦，获取其对应的索引，如果和弦未知则使用默认的未知和弦索引。 确保和弦索引在有效范围内。 根据和弦的顺序和位置，计算其在状态索引中的权重，并累加到总索引中。 确保最终索引在有效范围内。 返回计算得到的索引。
* 调用： get,
* 内部依赖描述：

### \_handle\_note\_release (service/MidiInput.py)

* 行号位置：115-119
* 重要性评分：2.50
* 功能描述： 处理释放音符，当音符在活动键列表中时，将其移除并触发键释放事件。
* 实现流程： 检查音符是否在活动键列表中。 如果音符在活动键列表中，将其移除。 触发键释放事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： remove,\_emit\_key\_release,
* 内部依赖描述：
  + remove: 该函数用于从数组中移除指定索引位置的元素，并返回该元素。
  + \_emit\_key\_release: 该函数用于触发按键释放事件，将MIDI音符转换为虚拟键值，并通过信号发出按键释放事件，同时记录日志并调用内部函数处理MIDI输入事件。

### load\_model (service/ChordHMM.py)

* 行号位置：255-269
* 重要性评分：2.50
* 功能描述： 该函数用于从指定路径加载模型参数，包括状态转移矩阵A、观测矩阵B、初始状态概率pi、和弦索引chord\_index等。
* 实现流程： 打开指定路径下的模型文件。 使用pickle模块加载模型数据。 将加载的数据分别赋值给类的属性A、B、pi、chord\_index、n\_states和n\_chords。 打印加载成功的消息。
* 引入包： pickle,utils.filePath,
* 调用： filePath,
* 内部依赖描述：
  + filePath: 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。

### \_handle\_note\_press (service/MidiInput.py)

* 行号位置：109-113
* 重要性评分：2.50
* 功能描述： 处理按下音符，如果音符不在活动键集合中，则将其添加到集合中，并触发音符按下事件。
* 实现流程： 检查音符是否在活动键集合中。 如果音符不在集合中，则将其添加到活动键集合中。 触发音符按下事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： add,\_emit\_key\_press,
* 内部依赖描述：
  + \_emit\_key\_press: 该函数用于触发按键按下事件，将MIDI音符转换为虚拟键码，并通过信号发出按键按下事件。同时，记录按键按下日志，并调用内部函数处理MIDI输入事件。

### build\_contexts (service/ChordHMMV2.py)

* 行号位置：53-66
* 重要性评分：2.40
* 功能描述： 该函数用于构建上下文，通过将情绪标签列表和风格标签组合成元组，并将这些元组添加到集合中，最终返回一个上下文的列表。这样可以确保每个上下文都是唯一的。
* 实现流程： 接收情绪标签列表的列表和风格标签列表作为参数。 遍历情绪标签列表和风格标签列表，使用zip函数将它们一一对应地组合。 将每个情绪标签列表转换为排序后的元组，以便进行哈希操作，确保情绪标签的顺序不影响上下文的唯一性。 将情绪标签元组和风格标签组合成一个新的元组，作为上下文。 将每个上下文添加到集合中，确保集合中的上下文是唯一的。 将集合转换为列表并返回，列表中的每个元素都是一个唯一的上下文。
* 引入包： time,pickle,utils.filePath,
* 调用： add,
* 内部依赖描述：

### onPressed (VirtualKeyboard.py)

* 行号位置：400-403
* 重要性评分：2.40
* 功能描述： 该函数用于处理按键事件，当按键被按下时，记录按键信息，并从队列中移除指定索引位置的元素。
* 实现流程： 记录按键信息：当按键被按下时，函数首先记录按键的索引，并使用logging.info输出按键信息。 检查队列长度：函数检查队列的长度是否大于按键的索引。 移除元素：如果队列长度大于按键索引，则调用remove函数从队列中移除指定索引位置的元素。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： info,remove,
* 内部依赖描述：
  + remove: 该函数用于从数组中移除指定索引位置的元素，并返回该元素。

### \_start\_keyboard\_listener (service/MidiInput.py)

* 行号位置：66-69
* 重要性评分：2.40
* 功能描述： 该函数用于启动一个键盘事件监听器，监听键盘按键的按下和释放事件，并在事件发生时调用相应的处理函数。
* 实现流程： 创建一个键盘监听器对象，监听按键按下和释放事件。 设置按键按下事件的处理函数为\_on\_key\_press。 设置按键释放事件的处理函数为\_on\_key\_release。 启动监听器，开始监听键盘事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： Listener,start,
* 内部依赖描述：

### open\_image\_window (VirtualKeyboard.py)

* 行号位置：250-253
* 重要性评分：2.40
* 功能描述： 该函数用于打开一个图像窗口。如果图像窗口尚未创建，则会创建一个新的MatrixView实例。然后，无论是否创建新实例，都会显示图像窗口。
* 实现流程： 检查图像窗口是否已经存在。 如果图像窗口不存在，则创建一个新的MatrixView实例。 显示图像窗口。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： MatrixView,show,
* 内部依赖描述：

### **init** (service/markov.py)

* 行号位置：7-29
* 重要性评分：2.30
* 功能描述： 该类用于初始化一个马尔科夫链模型，用于生成和弦序列。它接受一个和弦风格、和弦序列列表以及一个可选的阶数（默认为2）。类会将所有和弦汇总到一个列表中，并构建一个马尔科夫链，其中每个状态（由指定阶数的和弦组成）对应可能的下一个和弦。最后，它会记住最后一组状态和对应的下一个状态。
* 实现流程： 初始化一个空的和弦列表和一个空的马尔科夫链字典。 将所有和弦序列汇总到一个列表中。 遍历和弦列表，构建马尔科夫链。对于每个状态（由指定阶数的和弦组成），记录可能的下一个和弦。 记住最后一组状态和对应的下一个状态。 打印马尔科夫链的初始化结果。
* 引入包： random,

### stop\_timer (VirtualKeyboard.py)

* 行号位置：644-646
* 重要性评分：2.30
* 功能描述： 停止定时器和更新定时器
* 实现流程： 调用self.timer.stop()方法停止定时器 调用self.updateTimer.stop()方法停止更新定时器
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： stop,
* 内部依赖描述：
  + stop: 停止MIDI输入并清理相关资源。

### start\_timer (VirtualKeyboard.py)

* 行号位置：640-642
* 重要性评分：2.30
* 功能描述： 启动两个定时器，一个每秒触发一次，另一个每120毫秒触发一次。
* 实现流程： 调用self.timer.start(1000)，启动一个定时器，设置为每秒触发一次。 调用self.updateTimer.start(120)，启动另一个定时器，设置为每120毫秒触发一次。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： start,
* 内部依赖描述：

### index\_chords (service/numpyMarkov.py)

* 行号位置：18-29
* 重要性评分：2.20
* 功能描述： 该函数用于为和弦序列中的每个和弦分配一个唯一的索引，并返回一个和弦到索引的映射字典。同时，它还会为未知和弦分配一个独特的索引。
* 实现流程： 初始化一个空集合unique\_chords，用于存储唯一的和弦。 遍历输入的和弦序列sequences，将每个和弦添加到unique\_chords集合中，确保集合中只包含唯一的和弦。 计算unique\_chords集合的长度，并将其赋值给self.unknown\_chord\_index，作为未知和弦的唯一索引。 使用enumerate函数遍历unique\_chords集合，生成一个和弦到索引的映射字典，并返回该字典。
* 调用： update,
* 内部依赖描述：

### filePath (utils/filePath.py)

* 行号位置：3-4
* 重要性评分：2.20
* 功能描述： 该函数用于构建文件路径，通过将传入的文件目录与当前脚本所在目录的上两级目录进行拼接，返回完整的文件路径。
* 实现流程： 获取当前脚本所在目录的上两级目录。 将传入的文件目录与上两级目录进行拼接。 返回拼接后的完整文件路径。
* 引入包： os,
* 调用： dirname,
* 内部依赖描述：

### index\_chords (service/ChordHMM.py)

* 行号位置：29-38
* 重要性评分：2.00
* 功能描述： 该函数用于为和弦序列中的每个唯一和弦分配一个唯一的索引。
* 实现流程： 首先，创建一个空的集合unique\_chords用于存储唯一的和弦。 遍历输入的和弦序列列表sequences，将每个序列中的和弦更新到unique\_chords集合中，确保每个和弦只出现一次。 使用enumerate函数遍历unique\_chords集合，为每个和弦分配一个从0开始的唯一索引，并将和弦到索引的映射存储在一个字典中返回。
* 引入包： pickle,utils.filePath,
* 调用： update,
* 内部依赖描述：

### index\_chords (service/ChordHMMV2.py)

* 行号位置：42-51
* 重要性评分：2.00
* 功能描述： 该函数用于为和弦序列中的每个唯一和弦分配一个唯一的索引。
* 实现流程： 首先，创建一个空的集合unique\_chords，用于存储所有唯一的和弦。 遍历输入的和弦序列列表sequences，对于每个和弦序列seq，使用update方法将其中的和弦添加到unique\_chords集合中，确保每个和弦只出现一次。 最后，使用enumerate函数遍历unique\_chords集合，为每个和弦分配一个从0开始的唯一索引，并返回一个字典，其中键是和弦，值是对应的索引。
* 引入包： time,pickle,utils.filePath,
* 调用： update,
* 内部依赖描述：

### context\_to\_index (service/ChordHMMV2.py)

* 行号位置：90-99
* 重要性评分：2.00
* 功能描述： 该函数用于将情绪标签和风格标签转换为上下文索引。它首先对情绪标签进行排序并转换为元组，然后与风格标签一起组成上下文元组。最后，通过上下文元组在self.context\_index字典中查找对应的索引，如果存在则返回索引，否则返回-1。
* 实现流程： 接收情绪标签列表和风格标签作为输入参数。 对情绪标签列表进行排序并转换为元组。 将排序后的情绪元组与风格标签组合成上下文元组。 在self.context\_index字典中查找上下文元组对应的索引。 如果找到对应的索引，则返回该索引；否则返回-1。
* 引入包： time,pickle,utils.filePath,
* 调用： get,
* 内部依赖描述：

### midi\_notes\_to\_vector (service/ChordVectorFaiss.py)

* 行号位置：32-40
* 重要性评分：1.90
* 功能描述： 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。
* 实现流程： 初始化一个长度为 vector\_dim 的全零浮点数向量。 遍历输入的 MIDI 音符序列。 对于每个音符，如果其值在 0 到 vector\_dim 之间（包括 0 和 vector\_dim），则将向量中对应位置的值设为 1.0。 返回填充好的 One-Hot 编码向量。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： zeros,
* 内部依赖描述：

### chords\_to\_indices (service/ChordHMMV2.py)

* 行号位置：82-88
* 重要性评分：1.70
* 功能描述： 将和弦序列转换为对应的索引序列。
* 实现流程： 接收一个和弦列表作为输入。 遍历和弦列表中的每个和弦。 使用字典self.chord\_index查找每个和弦对应的索引。如果和弦不在字典中，则返回-1。 将查找得到的索引添加到结果列表中。 返回包含所有和弦索引的结果列表。
* 引入包： time,pickle,utils.filePath,
* 调用： get,
* 内部依赖描述：

### chords\_to\_indices (service/ChordHMM.py)

* 行号位置：53-59
* 重要性评分：1.70
* 功能描述： 将和弦序列转换为对应的索引序列。
* 实现流程： 接收一个和弦列表作为输入。 遍历和弦列表中的每个和弦。 使用self.chord\_index字典查找每个和弦对应的索引。如果和弦不在字典中，则返回-1。 将查找得到的索引添加到结果列表中。 返回包含所有和弦索引的结果列表。
* 引入包： pickle,utils.filePath,
* 调用： get,
* 内部依赖描述：

### transition\_matrix\_to\_string (service/numpyMarkov.py)

* 行号位置：110-116
* 重要性评分：1.70
* 功能描述： 将概率转移矩阵转换为字符串形式，便于查看和记录。
* 实现流程： 使用np.array2string方法将概率转移矩阵转换为字符串。 设置precision参数为2，确保输出的小数点后有两位。 使用separator参数为’, ’，设置元素之间的分隔符为逗号和空格。 返回转换后的字符串形式的转移矩阵。
* 调用： array2string,
* 内部依赖描述：

### set\_chord\_sequences (MatrixView.py)

* 行号位置：30-35
* 重要性评分：1.60
* 功能描述： 该函数用于设置新的和弦序列，并在和弦序列发生变化时更新图像。
* 实现流程： 检查传入的和弦序列是否与当前的和弦序列不同。 如果不同，则更新当前的和弦序列。 设置一个标志，表示图像需要更新。 调用更新图像的方法，以反映新的和弦序列。
* 引入包： sys,PyQt5.QtWidgets,PyQt5.QtCore,PyQt5.QtGui,service.numpyMarkov,io,
* 调用： update\_image,
* 内部依赖描述：
  + update\_image: 该函数用于根据当前和预测的和弦序列更新显示的图像，通过绘制和弦转换矩阵的热图来实现。

### closeEvent (VirtualKeyboard.py)

* 行号位置：566-571
* 重要性评分：1.60
* 功能描述： 处理窗口关闭事件，确保窗口在关闭前执行一些清理操作。
* 实现流程： 调用父类的closeEvent方法，确保父类的关闭逻辑得到执行。 调用self.close()方法，关闭当前窗口。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： closeEvent,
* 内部依赖描述：
  + closeEvent: 处理窗口关闭事件，确保窗口在关闭前执行一些清理操作。

### \_choose\_device (service/MidiInput.py)

* 行号位置：55-58
* 重要性评分：1.40
* 功能描述： 该函数用于手动选择MIDI设备。用户可以通过输入设备ID来指定设备，如果不输入则跳过使用键盘映射。
* 实现流程： 提示用户手动指定一个设备ID，或者按回车跳过使用键盘映射。 读取用户输入的设备ID。 如果用户输入了设备ID，则将其转换为整数并返回；如果用户按回车跳过，则返回None。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： strip,
* 内部依赖描述：

### **init** (utils/FileWorker.py)

* 行号位置：20-23
* 重要性评分：1.40
* 功能描述： 初始化一个和弦对象，将传入的和弦名称和按下的音符赋值给实例变量。
* 实现流程： 调用父类的初始化方法。 将传入的和弦名称赋值给实例变量 chord\_name。 将传入的按下的音符赋值给实例变量 pressing\_notes。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,utils.filePath,
* 调用： **init**,
* 内部依赖描述：
  + **init**: 该类用于初始化一个支持MIDI输入和键盘映射的系统，主要用于和弦序列预测，并支持情绪和风格标签作为上下文。它设置了一个窗口标题和大小，并初始化了一些变量来控制图像更新。类中还设置了标签来显示图像，并通过布局管理器将标签添加到窗口中。

### play\_note\_off (service/soundNoise.py)

* 行号位置：40-42
* 重要性评分：1.30
* 功能描述： 关闭指定音符
* 实现流程： 接收音符参数 调用fs对象的noteoff方法，关闭指定音符
* 引入包： platform,fluidsynth,utils.filePath,logging,
* 调用： noteoff,
* 内部依赖描述：

### play\_note\_on (service/soundNoise.py)

* 行号位置：36-38
* 重要性评分：1.30
* 功能描述： 该函数用于在乐器上播放指定音符，音符的音高由参数note决定，音量由参数velocity决定，默认音量为127。
* 实现流程： 接收音符的音高（note）和音量（velocity）作为参数，音量默认值为127。 调用乐器对象的noteon方法，传入通道号0、音符的音高和音量，以在乐器上播放指定音符。
* 引入包： platform,fluidsynth,utils.filePath,logging,
* 调用： noteon,
* 内部依赖描述：

### **init** (utils/musicUtils.py)

* 行号位置：8-10
* 重要性评分：1.30
* 功能描述： 初始化一个对象，并将传入的字符串赋值给实例变量pressing。
* 实现流程： 调用父类的初始化方法。 将传入的字符串参数赋值给实例变量pressing。
* 引入包： musicpy.algorithms,
* 调用： **init**,
* 内部依赖描述：
  + **init**: 该类用于初始化一个支持MIDI输入和键盘映射的系统，主要用于和弦序列预测，并支持情绪和风格标签作为上下文。它设置了一个窗口标题和大小，并初始化了一些变量来控制图像更新。类中还设置了标签来显示图像，并通过布局管理器将标签添加到窗口中。

### on\_virtual\_key\_pressed (VirtualKeyboard.py)

* 行号位置：649-650
* 重要性评分：1.20
* 功能描述： 当虚拟键盘上的某个键被按下时，调用虚拟键盘的虚拟键按下方法，并传递被按下的键的索引。
* 实现流程： 接收虚拟键盘上被按下的键的索引。 调用虚拟键盘的虚拟键按下方法，并将索引作为参数传递给该方法。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： virtual\_key\_pressed,
* 内部依赖描述：
  + virtual\_key\_pressed: 处理虚拟钢琴键盘按下事件，更新按键状态并触发相关操作。

### on\_virtual\_key\_released (VirtualKeyboard.py)

* 行号位置：653-654
* 重要性评分：1.20
* 功能描述： 当虚拟键盘上的某个键被释放时，调用虚拟键盘的虚拟键释放方法，并传递被释放键的索引。
* 实现流程： 接收虚拟键盘上某个键被释放的事件。 获取被释放键的索引。 调用虚拟键盘的虚拟键释放方法，并将索引作为参数传递给该方法。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： virtual\_key\_released,
* 内部依赖描述：
  + virtual\_key\_released: 处理虚拟钢琴键盘释放事件，更新按键状态并触发相关操作。

### getChordAttr (utils/musicUtils.py)

* 行号位置：15-16
* 重要性评分：1.20
* 功能描述： 该函数用于检测音乐中的和弦类型。
* 实现流程： 接收一个名为’pressing’的参数，该参数包含音乐的按键信息。 调用’musicpy.algorithms.detect’函数，传入’pressing’参数，并设置’get\_chord\_type’参数为True，以检测和弦类型。 返回检测到的和弦类型信息。
* 引入包： musicpy.algorithms,
* 调用： detect,
* 内部依赖描述：

### clear (utils/QueueUtil.py)

* 行号位置：32-33
* 重要性评分：1.20
* 功能描述： 该函数用于清空对象的数组属性。
* 实现流程： 调用对象的array属性的clear方法 清空array属性中的所有元素
* 调用： clear,
* 内部依赖描述：
  + clear: 该函数用于清空对象的数组属性。

### test\_something (utils/tests.py)

* 行号位置：7-8
* 重要性评分：1.20
* 功能描述： 该函数用于测试一个简单的断言，验证两个布尔值是否相等。
* 实现流程： 函数test\_something被调用。 执行断言self.assertEqual(True, True)，验证两个布尔值是否相等。 如果断言通过，测试通过；如果断言失败，测试失败并抛出异常。
* 引入包： unittest,service.numpyMarkov,
* 调用： assertEqual,
* 内部依赖描述：

### getNormalChord (utils/musicUtils.py)

* 行号位置：12-13
* 重要性评分：1.20
* 功能描述： 该函数用于检测音乐中的正常和弦。
* 实现流程： 调用 musicpy 库中的 algorithms 模块的 detect 函数。 传入 self.pressing 作为参数，该参数可能包含音乐的按键信息。 detect 函数分析按键信息，识别出音乐中的正常和弦。 返回识别出的正常和弦结果。
* 引入包： musicpy.algorithms,
* 调用： detect,
* 内部依赖描述：

### print\_model\_parameters (service/ChordHMM.py)

* 行号位置：226-235
* 重要性评分：1.00
* 功能描述： 该函数用于打印隐马尔可夫模型（HMM）的参数，包括状态转移矩阵（A）、发射矩阵（B）和初始状态概率（pi）。
* 实现流程： 打印状态转移矩阵（A） 打印发射矩阵（B） 打印初始状态概率（pi）
* 引入包： pickle,utils.filePath,

### print\_model\_parameters (service/ChordHMMV2.py)

* 行号位置：304-313
* 重要性评分：1.00
* 功能描述： 该函数用于打印隐马尔可夫模型（HMM）的参数，包括状态转移矩阵（A）、发射矩阵（B）和初始状态概率（pi）。
* 实现流程： 打印状态转移矩阵（A） 打印发射矩阵（B） 打印初始状态概率（pi）
* 引入包： time,pickle,utils.filePath,

### midi\_notes\_to\_vector (service/ChordVectorAnnoy.py)

* 行号位置：27-35
* 重要性评分：0.90
* 功能描述： 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。
* 实现流程： 初始化一个长度为 vector\_dim 的全零向量。 遍历输入的 MIDI 音符序列。 对于每个音符，如果其值在 0 到 vector\_dim 之间（包括 0 和 vector\_dim），则将该音符对应的向量位置设为 1。 返回转换后的向量向量。
* 引入包： os,pickle,annoy,utils.filePath,

### getRoot (utils/StringUtils.py)

* 行号位置：7-12
* 重要性评分：0.60
* 功能描述： 该函数用于获取字符串的根节点。根节点由字符串的第一个字符组成，如果第二个字符是’b’或’#’，则将第二个字符也包含在根节点中。
* 实现流程： 初始化一个空字符串root。 将字符串self.str的第一个字符添加到root中。 检查字符串self.str的第二个字符是否为’b’或’#’。 如果是，则将第二个字符也添加到root中。 返回最终的root字符串。

### length (utils/QueueUtil.py)

* 行号位置：5-9
* 重要性评分：0.50
* 功能描述： 该函数用于计算数组的长度。如果数组为空，则返回0；否则返回数组的长度。
* 实现流程： 检查数组是否为空 如果数组为空，返回0 如果数组不为空，使用len函数计算数组的长度并返回

### **init** (service/MatchingScales.py)

* 行号位置：36-39
* 重要性评分：0.40
* 功能描述： 初始化一个音乐分析对象，设置根音符、输入音符列表和阈值。
* 实现流程： 接收根音符、输入音符列表和阈值作为参数。 将根音符、输入音符列表和阈值分别赋值给对象的属性root\_note、input\_notes和threshold。 完成对象的初始化。

### last (utils/QueueUtil.py)

* 行号位置：35-37
* 重要性评分：0.30
* 功能描述： 该函数用于获取数组的最后一个元素。如果数组长度不为0，则返回数组的最后一个元素；否则，不执行任何操作。
* 实现流程： 检查数组的长度是否为0。 如果数组长度不为0，则返回数组的最后一个元素。 如果数组长度为0，则不执行任何操作。

### pop (utils/QueueUtil.py)

* 行号位置：17-18
* 重要性评分：0.20
* 功能描述： 该函数用于从数组的头部移除并返回第一个元素。
* 实现流程： 调用数组的pop方法，并传入参数0，表示移除数组的第一个元素。 返回被移除的元素。

### push (utils/QueueUtil.py)

* 行号位置：14-15
* 重要性评分：0.20
* 功能描述： 该函数用于将一个值添加到数组的末尾。
* 实现流程： 接收一个值作为参数。 将该值追加到数组的末尾。

### is\_Empty (utils/QueueUtil.py)

* 行号位置：11-12
* 重要性评分：0.20
* 功能描述： 检查数组是否为空
* 实现流程： 获取当前对象的array属性 判断array属性是否为None 如果array属性为None，则返回True，表示数组为空；否则返回False，表示数组不为空

### top (utils/QueueUtil.py)

* 行号位置：20-21
* 重要性评分：0.20
* 功能描述： 获取堆顶元素
* 实现流程： 从堆的数组中返回第一个元素，即堆顶元素

### index (utils/QueueUtil.py)

* 行号位置：26-27
* 重要性评分：0.20
* 功能描述： 该函数用于从数组中获取指定索引位置的元素。
* 实现流程： 接收一个整数参数 i，表示要获取的元素的索引位置。 使用索引 i 从数组 self.array 中获取对应的元素。 返回获取到的元素。

### **init** (utils/QueueUtil.py)

* 行号位置：2-3
* 重要性评分：0.20
* 功能描述： 初始化一个类实例，该实例可以存储一个数组。
* 实现流程： 定义一个名为\_\_init\_\_的初始化方法，该方法接受一个可选参数array，默认值为None。 在方法内部，将传入的array参数赋值给实例变量self.array。

### travel (utils/QueueUtil.py)

* 行号位置：23-24
* 重要性评分：0.20
* 功能描述： 该函数用于打印对象的数组属性。
* 实现流程： 调用函数travel。 函数内部打印对象的array属性。

### remove (utils/QueueUtil.py)

* 行号位置：29-30
* 重要性评分：0.20
* 功能描述： 该函数用于从数组中移除指定索引位置的元素，并返回该元素。
* 实现流程： 接收一个索引参数。 使用数组的pop方法，根据索引移除元素。 返回被移除的元素。

### **init** (utils/StringUtils.py)

* 行号位置：4-5
* 重要性评分：0.20
* 功能描述： 初始化一个字符串对象，将传入的字符串赋值给实例变量str。
* 实现流程： 接收一个字符串参数。 将接收到的字符串赋值给实例变量str。

### getType (utils/StringUtils.py)

* 行号位置：14-15
* 重要性评分：0.20
* 功能描述： 该函数用于获取一个包含特定类型字符串的列表。
* 实现流程： 定义一个名为type的列表，其中包含四个字符串元素：‘minor’, ‘major’, ‘m7’, ‘m9’。 返回这个列表作为函数的结果。

### initializeFluidsynth (utils/SoundTest.py)

* 行号位置：1-23
* 功能描述： 初始化 Fluidsynth 并加载 SoundFont

### (.gitignore)

### ChordCrafter (ChordCrafter.spec)

### ChordCrafter\_win64 (ChordCrafter\_win64.spec)

### LICENSE (LICENSE)

### README (README.md)

### README\_EN (README\_EN.md)

### dorian (labels/dorian.model)

### jazz (labels/jazz.model)

### pop (labels/pop.model)

### rock (labels/rock.model)

### 五度圈模型 (labels/五度圈模型.model)

### 常用终止式 (labels/常用终止式.model)

### 忧郁 (labels/忧郁.model)

### 恢弘 (labels/恢弘.model)

### 我的自制数据集 (labels/我的自制数据集.model)

### pressing\_chord\_mappings (models/pressing\_chord\_mappings.cache)

### 多利亚宇 (records/多利亚宇.model)

### 悲伤爵士 (records/悲伤爵士.model)

### 我的自制和弦 (records/我的自制和弦.model)

### 马里奥终止式 (records/马里奥终止式.model)

### requirements (requirements.txt)

### **init** (service/**init**.py)

### put\_file\_to\_here (sounds/FluidR3\_GM/put\_file\_to\_here)

### **init** (utils/**init**.py)