

一种自动化鸟类声学特征提取框架： 以深港边境数据为例

An Automated Framework for Extracting Acoustic Features of Birds:

Using Data from the Shenzhen-Hong Kong Border as an Example

陈沐 24327004 物理（理学院）

关键词 (Keywords):

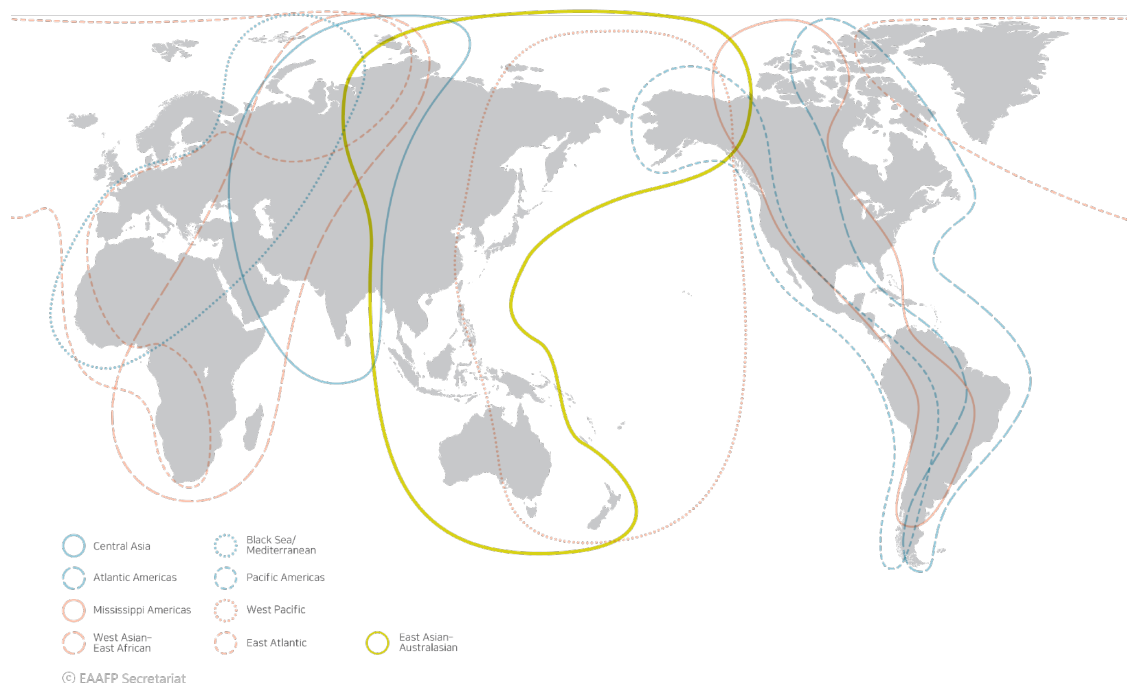
自动化数据采集 (Automated Data Acquisition), 生态声学 (Eco-acoustics), 被动声学监测 (Passive Acoustic Monitoring), 生物多样性 (Biodiversity), 鸟类鸣声 (Avian Vocalization), 特征工程 (Feature Engineering), Wolfram Mathematica, 深港跨境保护 (Shenzhen-Hong Kong Cross-border Conservation), 梅尔频谱图 (Mel Spectrogram), Wolfram Mathematica

摘要 Abstract

本报告阐述了一套为深港边境这一独特生态系统设计的、从数据采集到特征生成的端到端自动化流水线。基于Wolfram Mathematica语言，集成了一个高效的网络数据采集工具（爬虫），以及一套高度鲁棒的、事件驱动的声学特征工程方法，用于将原始音频转化为标准化的梅尔频谱图“声学指纹”，并导出为可复用的224*224 Pixels PNG频谱文件。本研究的核心贡献在于成功构建了一个可复现、可扩展的技术框架，能够自动采集高质量的指定鸟类音频数据；并将海量的、非结构化的网络鸟鸣音频高效处理，把声音问题转化成了图像问题，可匹配图像神经网络的输入层，将数据输出为适用于专家与先进机器学习分析的、高质量的结构化特征数据。我们强调，该自动化方法在建立长期、大规模、标准化的生物多样性数据库中具有革命性的作用，具有强大的泛化应用能力，并为未来的跨区生态协同研究提供了坚实的数据基础。

第一章：引言 (Introduction)

1.1 研究背景与生态学意义



浅绿框中为“东亚-澳大利西亚迁飞区”¹

深港边境地区，特别是以米埔自然保护区为核心的湿地群，构成了全球尺度下东亚 - 澳大利西亚候鸟迁徙路线上一个至关重要的生态节点。² 鸟类作为对环境变化高度敏感的指示物种，其种群多样性与丰度的动态能有效反映该区域生态系统的健康状况。然而，这一区域正面临着由高度城市化进程所带来的独特压力，其核心矛盾在于传统基塘渔业衰落后，大量经济效益低下的荒废鱼塘，却意外成为了具有极高生态价值的人工湿地，为迁徙水鸟提供了关键的觅食地。因此，如何在经济发展与生态保护之间取得平衡，科学评估并管理这些土地，是区域可持续发展的核心挑战。传统的依赖人力的鸟类调查方法，因成本高昂、时空覆盖度有限（尤其无法覆盖禁区和夜间），且易受观察者主观偏差影响，已难以满足对此复杂系统进行精细化、长期化监测的需求。



米埔自然护理区被誉为香港的「雀鸟天堂」，是白眉鸭和凤头潜鸭等候鸟的重要栖息地。每年约有60,000只水鸟于米埔渡冬，形成壮观的场面。自1995年起，香港米埔内后海湾正式根据《拉姆萨尔公约》列为「国际重要湿地」。

米埔自然护理区除了在鸟类保育发挥重要作用，当中具高保育价值的欧亚水獭只在米埔内后海湾拉姆萨尔湿地及附近出没。

通过积极管理，米埔自然护理区维护着多样化的湿地生境，包括基围、红树林、泥滩、淡水池塘和芦苇丛，为不同的野生动物提供了多样化的栖息地，并保育和促进本地生物多样性。³

1.2 声学监测：生态学研究的新范式

被动声学监测（Passive Acoustic Monitoring, PAM）作为一种非侵入性的新兴技术，能够实现全天候、长期、大规模的自动化数据采集，为克服传统方法的局限性提供了革命性的解决方案。然而，PAM产生了海量的原始音频数据，如何从中高效、准确地提取出有价值的生物学信息，是当前生态声学领域面临的核心技术瓶颈。数据处理的效率和质量，直接决定了声学监测能否从一个数据采集工具，转变为一个强大的科学分析与保护决策支持系统。⁴

1.3 研究目标与核心问题

本研究的目标是构建并验证一套从原始鸟鸣数据采集到机器学习可用特征提取的全自动化、高鲁棒性工作流。我们旨在设计一个标准化的技术流程，以克服原始声学数据在来源、质量和内

容上的异构性与复杂性，从而为深港跨境生态研究提供一个可靠、可比较、可扩展的数据基础。

第二章：数据采集与管理：自动化鸟鸣数据库的构建

2.1 数据源：Xeno-Canto 全球鸟鸣数据库



xeno-canto
Sharing wildlife sounds from around the world

Search recordings... Search

About Explore Upload Sounds Forum Mysteries Articles

20 years of XC!

XC178537 0:00 2:02:50
黑背信天翁 *Phoebeastria immutabilis* · call, clattering bills
david m

Xeno-canto was launched on May 30, 2005. Twenty years ago! And it is still alive and kicking. We've written [a short article](#) on those years. Have a read, and perhaps, by way of a present, comment with your favourite XC experience. And yes, we are keen on birthday presents! What better day than to start [donating](#) to XC. Any gift helps us not only to continue, but it also helps us to send the message that open data is not free: that it involves time and money, and that there is a community that realises and values that.

What is xeno-canto?

xeno-canto is a website dedicated to sharing wildlife sounds from all over the world. Whether you are a research scientist, a birder, or simply curious about a sound that you heard out your kitchen window, we invite you to listen, download, and explore the wildlife sound recordings in the collection.

But xeno-canto is more than just a collection of recordings. It is also a collaborative project. We invite you to share your own recordings, help identify mystery recordings, or share your expertise in the forums. Welcome!

Collection Statistics

Collection	Statistics
1007580	R
980528	IC
827279	B
12996	Sj
12894	Si
11985	R
18552:55:10	R
900530	R

[More...](#)

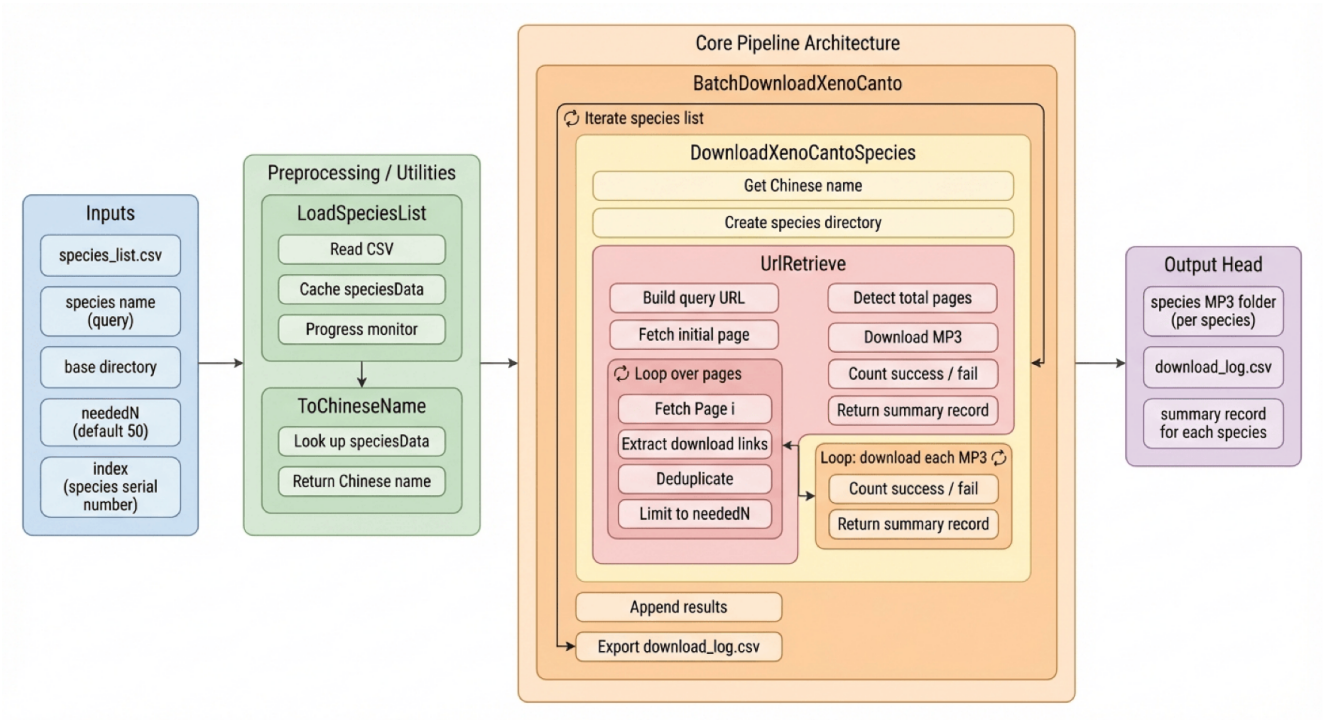
Latest New S

- Blue Wildebeest
- Hylodes fredii
- Lesser Asian Y
- Black-bearded
- David's Myotis

[More...](#)

我们选择了全球最大、最活跃的鸟鸣声学分享平台 Xeno-Canto作为我们的数据源。该平台由全球的公民科学家和专业研究者共同贡献，积累了海量的、带有物种标签和地理位置信息的鸟鸣录音，是现代鸟类学研究不可或缺的宝贵资源。

2.2 基于 Mathematica 的自动化数据爬虫



函数名 (Function Name)	核心作用 (Core Purpose)	输入 (Inputs)	返回值 (Return Value)
LoadSpeciesList	加载并缓存物种列表： 从指定的 CSV 文件中读取物种信息（ 中英文名等），并将 其存入内存以备后续 快速查询。	key (CSV 文件路径), baseDir (可选的基础目录)	Integer (加载的记录总 或 \$Failed)
ToChineseName	翻译物种名称： 根据已加载的物种列表 ，将输入的英文物种 名查询并转换为对应 的中文名。	name (英文物种名)	String (对应的中文 或 \$Failed)
UrlRetrieve	抓取下载链接： 针对单个物种，访问 Xeno-Canto 网站，自动识别并遍 历所有分页，抓取指 定数量的音频文件下 载链接。	key (物种名), neededN (可选, 所需链接数量)	List (包含多个下载 接 URL 字符串的列表)
DownloadXenoCantoSpecies	下载单个物种音频：为单 个物种创建规范的文 件夹，调用 UrlRetrieve 获取链接， 然后逐一下载音频文件 ，并处理下载过程中 的成功与失败。	index, species, baseDir, neededN (物种编号、名称 、基础目录、 下载数量)	Association (包含下载任务 要的关联， 用于日志记录)
BatchDownloadXenoCanto	批量处理与日志记录： 作为顶层调度函数， 它读取完整的物种列表 ，依次为每个物种执 行下载任务， 并在全部完成后，将 所有结果汇总并导出 为一份 CSV 日志文件。	csvPath, baseDir, neededN (物种列表路径、 基础目录、 下载数量)	String (最终生成的日 文件的完整路 径)

```
In[*]:= (* ===== *)
(*Preserve Space for 中英文名转换函数*)
(* ===== *)
(* ===== *)
(*Module 1:LoadSpeciesList*)
```

```

(*功能：导入 species_list.csv 并缓存结果（含进度条）*)
(* =====*)
ClearAll[LoadSpeciesList];
LoadSpeciesList[key_String,
  baseDir_ : FileNameJoin[{$UserDocumentsDirectory, "XenoCantoDownloads"}]] :=
Module[{csvPath, raw, total}, (*拼接 CSV 文件完整路径*)
  csvPath = If[FileExistsQ[key], key, FileNameJoin[{baseDir, key}]];
  (*检查文件存在性*)
  If[! FileExistsQ[csvPath], Print[" CSV 文件不存在: ", csvPath];
    Return[$Failed];];
  Print[" 正在加载物种表, 请稍候..."];
  (*导入 CSV*)raw = Import[csvPath, {"CSV", "Data"}];
  total = Length[raw] - 1;
  (*减去表头*) (*带进度条导入数据*)
  Monitor[ToChineseName`speciesData = Table[Pause[0.001];
    (*模拟进度条更新*)raw[[i]], {i, 2, Length[raw]}],
    ProgressIndicator[i, {2, Length[raw]}]];
  Print[" 已加载物种表, 共 ", total, " 条记录。"];
  Length[ToChineseName`speciesData]]

(* =====*)
(*Module 2:ToChineseName*)
(*功能：在缓存数据中查找中文名*)
(* =====*)

ClearAll[ToChineseName];
ToChineseName[name_String] :=
Module[{match}, If[! ValueQ[ToChineseName`speciesData],
  Print[" 请先运行 LoadSpeciesList[] 载入物种列表。"];
  Return[$Failed];];
  match = SelectFirst[ToChineseName`speciesData,
    StringTrim[#[[2]]] == StringTrim[name] &];
  If[match === Missing["NotFound"], Print[" 未找到物种: ", name];
    $Failed, match[[3]]]]

(* =====*)
(*Function:UrlRetrieve-获取下载链接列表*)
(* =====*)
UrlRetrieve[key_, neededN_ : 50] :=
Module[{pageCount, urlList = {}, i, keyStr, xml, allStrings, downloadLinks,
  currentUrl}, keyStr = StringReplace[ToString[key], " " -> "+"];

```



```

currentUrl = StringJoin[baseUrl, keyStr, baseUrlFirstSuffix];
Print["  Fetching initial page for ", key, " ..."];
Quiet[Check[xml = Import[currentUrl, "XMLObject"];
  allStrings = Cases[xml, _String, Infinity];
  pageCount = Max@ToExpression@StringDrop[Flatten[
    StringCases[allStrings, "pg=" ~~ NumberString]], 3], pageCount = 1]];

If[pageCount < 0, pageCount = 1, {}, {}];

Print["  Total pages found: ", pageCount];
i = 1;
While[i ≤ pageCount && Length[urlList] < neededN,
  currentUrl = StringJoin[baseUrl, keyStr, baseUrlSuffix, ToString[i]];
  Print["  Fetching page ", i, " ..."];
  Quiet[Check[xml = Import[currentUrl, "XMLObject"];
    allStrings = Cases[xml, _String, Infinity];
    downloadLinks = Flatten[StringCases[allStrings,
      "https://xeno-canto.org/" ~~ NumberString ~~ "/download"]];
    urlList = DeleteDuplicates@Join[urlList, downloadLinks];
    Print["  Error fetching page ", i];
    Continue[[]];];
  Pause[RandomReal[{1.0, 2.5}]];
  i++];
Print["  Retrieved ", Length[urlList], " valid download links."];

(*Print[pageCount];*)
Take[urlList, UpTo[neededN]]];

(* =====*)
(*Function:DownloadXenoCantoSpecies*)
(* =====*)
DownloadXenoCantoSpecies[
  index_Integer, species_String, baseDir_, neededN_:50] :=
Module[{speciesDir, urls, total, successCount = 0, failCount = 0, fileName,
  targetFile, chineseName}, chineseName = ToChineseName[species];
speciesDir = FileNameJoin[{baseDir, StringJoin[ToString[index], " ",
  StringReplace[species, " " → "_"], " (" , chineseName, ")"]]];
If[! DirectoryQ[speciesDir],
  CreateDirectory[speciesDir, CreateIntermediateDirectories → True]];
urls = UrlRetrieve[species, neededN];
total = Length[urls];
Print["  Start downloading ", total,

```

```

    " files for ", species, " (" , chineseName, ") ...");
Do[fileName = StringReplace[
    StringCases[url, "https://xeno-canto.org/" ~~ id : NumberString ~~
        "/download" :> {id <> ".mp3"}][[1], ".mp3.mp3" -> ".mp3"];
targetFile = FileNameJoin[{speciesDir, fileName}];
Quiet[Check[URLDownload[url, targetFile];
    successCount++;
    Print["    [" , successCount, "/", total, "] ", fileName];, failCount++;
    Print["    Failed: ", url];
    Continue[]];];
Pause[RandomReal[{1.0, 2.5}]];, {url, urls}];
Print["    Done! ", successCount, " / ", total,
    " files downloaded successfully, ", failCount, " failed."];
<|"No" -> index, "Species Name" -> species, "Chinese Name" -> chineseName,
    "Total Links" -> total, "Success Count" -> successCount,
    "Fail Count" -> failCount, "Timestamp" -> DateString[
        {"Year", "-", "Month", "-", "Day", " ", "Hour", ":", "Minute"}] |>];

(* =====*)
(*Function:BatchDownloadXenoCanto+Logging*)
(* =====*)
BatchDownloadXenoCanto[csvPath_, baseDir_, neededN_ : 50] :=
Module[{speciesList, total, results = {}, row, logFile},
    If[! FileExistsQ[csvPath], Print["    CSV file not found: ", csvPath];
        Return[$Failed];];
    speciesList = Import[csvPath];
    (*去掉第一行表头*)
    speciesList = Rest[speciesList];
    total = Length[speciesList];
    Print["    Loaded ", total, " species from CSV."];
    Do[row = speciesList[[i];
        Print["\n====="];
        Print["    Processing [" ,
            row[[1], "/", total, "]: " , row[[2], " (" , row[[3], ")"]];
        Print["====="];
        AppendTo[results,
            Quiet[Check[DownloadXenoCantoSpecies[row[[1], row[[2], baseDir, neededN], <|
                "No" -> row[[1], "Species Name" -> row[[2], "Chinese Name" -> row[[3],
                "Total Links" -> 0, "Success Count" -> 0, "Fail Count" -> "Error",
                "Timestamp" -> DateString[] |>]]];, {i, 1, total}];
    (*保存日志*) logFile = FileNameJoin[{baseDir, "download_log.csv"}];

```

```
Export[logFile, Prepend[results /. a_Association -> Values[a],
  {"No", "Species Name", "Chinese Name", "Total Links",
   "Success Count", "Fail Count", "Timestamp"}]];
Print[" All species processed! Log saved at: ", logFile];
logFile];
```

```
(* =====*)
(*基础配置*)
(* =====*)
baseUrl = "https://xeno-canto.org/explore?query=";
baseUrlSuffix = "&dir=0&order=qual&view=0&pg=";
baseUrlFirstSuffix = "&dir=0&order=qual&view=0";
```

```
In[* ]:= (* =====*)
(*使用 species_list_for_download.csv*)
(* =====*)
baseDir = FileNameJoin[{$UserDocumentsDirectory, "XenoCantoDownloads"}];
csvFilePath = FileNameJoin[{baseDir, "species_list_for_download.csv"}];

(*CSV 文件表头: No,Species Name,Chinese Name*)
LoadSpeciesList["species_list_for_download.csv", baseDir];
Print[" Testing Translation Module: \n Chinese Penduline Tit - ",
  ToChineseName["Chinese Penduline Tit"]];
```

```
If[!DirectoryQ[baseDir],
  CreateDirectory[baseDir, CreateIntermediateDirectories -> True];
Print[" Directory created: ", baseDir],
Print[" Directory already exists: ", baseDir]];
```

正在加载物种表, 请稍候...

已加载物种表, 共 9 条记录。

Testing Translation Module:

Chinese Penduline Tit - 中华攀雀

Directory already exists: /PathToUserDocuments/XenoCantoDownloads

Usage:

```
In[* ]:=
BatchDownloadXenoCanto[csvFilePath, baseDir, 50];
```

```

Loaded 9 species from CSV.

=====

Processing [1/9]: Plain Prinia (褐头鹟莺)

=====

Fetching initial page for Plain Prinia ...
Total pages found: 10
Fetching page 1 ...
Fetching page 2 ...
Retrieved 60 valid download links.
Start downloading 50 files for Plain Prinia (褐头鹟莺) ...
[1/50] 1040996.mp3
[2/50] 1040995.mp3
[3/50] 1030827.mp3
...
Done! 50 / 50 files downloaded successfully, 0 failed.
All species processed! Log saved at:
/.../Documents/XenoCantoDownloads/download_log.csv

```

2.3 AutoDownloader 五个核心亮点

亮点一：基于符号表达式的智能 Web 解析与高效数据规整

该爬虫最核心的革命性亮点，在于其摒弃了传统的、基于正则表达式的脆弱文本解析方式，转而采用 Mathematica 独有的符号式文档处理技术，并结合了高效的数据规整策略，从根本上提升了数据提取的鲁棒性与效率。传统爬虫依赖对 HTML 源码进行文本匹配，极易因网站前端结构的微小变动而失效。AutoDownloader 则通过 `xml = Import[currentUrl, "XMLObject"]` 将网页直接转化为一个结构化的符号表达式树。这一步的革命性在于，它将非结构化的网页文档，变成了可供程序逻辑直接查询和遍历的结构化对象。

在此基础上，该工具的“智能”体现在其高效且目标明确的链接获取策略中。

首先，它通过精心构建的 URL (`baseUrlFirstSuffix = "...&order=qual..."`)，直接请求 Xeno-Canto 服务器返回按“质量”排序的结果，这是一种巧妙的“借力”，将排序的计算负担交给了优化过的后端服务器，确保了获取到的链接从一开始就是最高质量的。

其次，`UrlRetrieve` 函数中的 `While[i <= pageCount && Length[urlList] < neededN]` 循环逻辑，构建了一个智能“刹车”机制。它不会盲目地抓取所有页面，而是持续监控已获取的链接数量，一旦满足用户定义的 `neededN` 需求，便会立即停止后续的网络请求，极大地节省了时间和网络资源，这对于大规模数据采集至关重要。

最后，在数据提取的最后一环，`Flatten[StringCases[allStrings, ...]]` 的应用堪称点睛

之笔。`StringCases` 在执行模式匹配时，其返回结果可能是一个嵌套列表（例如 `{{link1}, {link2}}`），这不利于直接处理。`Flatten[]` 函数以一种极其简洁和高效的方式，将这种不规则的嵌套结构“压平”，转化为一个干净、统一的一维列表 `{link1, link2, ...}`。这一步看似微小，却体现了 Mathematica 在数据“清洗”和“规整”方面的强大能力，它确保了无论原始数据结构多么复杂，最终进入下载流程的都是格式统一、立即可用的纯净数据，为整个自动化流程的稳定性提供了坚实保障。

亮点二：基于通用文本分析的高度泛化与迁移潜力

尽管 `AutoDownloader` 当前的应用场景是下载 Xeno-Canto 的鸟鸣音频，但其底层设计，特别是其核心的文本分析与模式匹配逻辑，具有极高的通用性和可泛化性，使其能够以最小的修改成本，被迅速迁移到其他截然不同的数据抓取任务中。这体现了 Mathematica 语言在符号处理和模式识别方面的深厚功力，也揭示了该项目作为一种“通用数据采集框架”的巨大潜力。

其通用性的基石在于 `UrlRetrieve` 函数中数据提取的核心逻辑：`allStrings = Cases[xml, _String, Infinity];` 配合 `downloadLinks = Flatten[StringCases[allStrings, pattern]]`。这个组合的强大之处在于，它将“抓取”这个动作抽象成了两个可配置的步骤：

- **1. URL 模板定义：**爬取的目标由 `baseUrl` 和 `baseUrlSuffix` 等几个字符串变量定义。要切换到另一个网站，例如一个在线图书馆，只需将这几个变量修改为该图书馆的搜索 URL 格式即可。
- **2. 目标模式定义：**需要从页面中提取何种信息，完全由 `StringCases` 中的 `pattern` 决定。在当前代码中，这个模式是 `"https://xeno-canto.org/" ~~ NumberString ~~ "/download"`，用于匹配音频下载链接。

现在，设想一个全新的任务：我们需要从一个学术论文网站上，根据作者姓名批量下载其所有论文的 PDF 链接。迁移 `AutoDownloader` 的过程将异常简单：

- 修改 `baseUrl` 等变量，使其指向该学术网站的搜索接口。
- 修改 `UrlRetrieve` 函数中 `StringCases` 的模式。我们只需观察目标网站 PDF 链接的格式，例如，它可能是 `"https://academic-site.com/papers/" ~~ __ ~~ ".pdf"`。将这个新模式替换掉原来的模式，整个爬虫的核心逻辑就完成了迁移。

甚至，我们可以提取不止一种信息。只需在 `UrlRetrieve` 中增加几行 `StringCases` 代码，就可以用不同的模式同时提取论文标题、发表年份、DOI 号等多种元数据。由于底层依赖的是对 `Import[... , "XMLObject"]` 符号树的遍历和模式匹配，这种方法对于目标网站的 HTML 结构变化具有很强的适应性。无论是链接在一个 `<a>` 标签里，还是作为一个 `<button>` 的 `onclick` 事件参数，只要其最终表现为文本形式，`Cases` 和 `StringCases` 就能大概率地将其捕获。

这种将“做什么”（由 URL 和模式定义）与“怎么做”（由 `Import`, `Cases`, `Check`, `Pause` 等组成的健壮执行框架）分离的设计，使得 `AutoDownloader` 的核心引擎具备了“即插即用”的能力。它不再是一个针对特定网站的爬虫，而是一个通用的、基于文本模式分析的自动化数据采集解决方案，能够被快速应用于文献计量、市场情报收集、社交媒体分析等众多领域，展现了其作为大规模数据处理工具的革命性潜力。

亮点三：双层级联的“堡垒式”异常捕获架构

在真实世界的网络环境中，自动化任务的连续性和稳定性是最大的挑战。`AutoDownloader` 通过一套精心设计的、双层级联的“堡垒式”异常捕获架构，构建了一个极其健壮的执行引擎，确

保了其在面对网络波动、服务器故障甚至程序逻辑错误时，依然能够最大限度地完成任务。这种架构的革命性在于，它为自动化流程提供了从“微观”到“宏观”的全方位保护。

第一层，是位于 `DownloadXenoCantoSpecies` 函数内部的“精细化”异常捕获。这里的 `Quiet[Check[URLDownload[url, targetFile];...]]` 结构是系统的第一道防线。它针对的是最小的操作单元——单个文件的下载。如果某个 URL 因为链接失效、网络超时或服务器返回错误而下载失败，`Check` 会精准地捕获这个异常，执行 `failCount++` 并打印失败日志，然后通过 `Continue[]` 指令，优雅地跳过当前失败的循环，继续尝试下载列表中的下一个文件。这确保了单个文件的失败，绝对不会影响到同一物种下其他有效文件的下载，实现了故障的最小化隔离。

第二层，是位于顶层入口函数 `BatchDownloadXenoCanto` 中的“守护式”异常捕获。代码 `AppendTo[results, Quiet[Check[DownloadXenoCantoSpecies[...], ...]]]` 构建了系统的第二道，也是最重要的一道防线。它的保护对象是整个 `DownloadXenoCantoSpecies` 函数的执行过程，即针对“单个物种”的完整处理流程。试想，如果因为某种罕见原因（如物种名包含非法字符导致创建目录失败），使得 `DownloadXenoCantoSpecies` 函数内部发生了无法被第一层 `Check` 捕获的致命错误，导致整个函数执行失败。如果没有这第二层保护，整个批量下载任务就会在此处崩溃。而现在，这个外层的 `Check` 会捕获到整个函数的失败，并执行预设的备用方案——记录一个包含错误信息的 `Association` 到结果列表中。然后，主循环会继续处理 CSV 文件中的下一个物种。这种双层嵌套的错误处理机制，确保了即使内部模块发生严重故障，也绝不会危及主任务的继续航行，这是实现真正“无人值守”、可长期稳定运行的大规模自动化系统的核心技术基石。

亮点四：数据与逻辑分离的高度可扩展架构

`AutoDownloader` 的设计精髓之一在于其彻底贯彻了“数据与逻辑分离”的核心软件工程原则，这使其不仅仅是一个固化的下载工具，而是一个高度可扩展、易于维护的自动化数据处理框架。其革命性在于，它将任务的具体“内容”（即要下载哪些物种）从程序的“行为”（即如何下载）中完全剥离，使得非技术人员也能轻松管理和扩展任务范围。

这一架构的核心体现在 `BatchDownloadXenoCanto` 函数与外部 `species_list_for_download.csv` 文件的交互上。整个爬虫的所有工作，都由这个简单的 CSV 文件驱动。文件中每一行定义了一个独立的任务单元（一个物种），包含了任务所需的所有元数据（编号、物种名、中文名）。当 `BatchDownloadXenoCanto` 启动时，它首先 `Import[csvPath]`，将这个外部数据源加载到内存中，然后遍历这个列表，依次为每个物种执行下载流程。这种设计的优势是巨大的：

- **1. 极佳的可扩展性：**当研究需求变化，需要增加或减少下载的物种时，用户无需触碰任何一行 Mathematica 代码。他们只需要用 Excel 或任何文本编辑器打开 `species_list_for_download.csv` 文件，增删或修改其中的行即可。这使得任务的管理权限从程序员下放到了数据管理者或研究人员手中，极大地提升了工作效率和灵活性。
- **2. 任务的可中断与恢复：**对于大规模下载任务，一次性完成所有下载是不现实的。由于任务列表是外部化的，用户可以轻松地将一个包含数千个物种的大列表，拆分成多个小列表分批执行。或者，在任务意外中断后，可以简单地编辑 CSV 文件，删除已成功下载的行，然后重新运行脚本，即可实现任务的断点续传。

- **3. 代码的通用性：**`DownloadXenoCantoSpecies` 和 `UrlRetrieve` 等核心函数被设计为通用的、接受物种名作为参数的“黑盒”。它们不关心这个物种名是从哪里来的，只负责完成自己的任务。这种设计使得代码逻辑高度内聚，易于测试和复用。

总而言之，这种数据驱动的架构，将 `AutoDownloader` 从一个“一次性脚本”提升到了一个“可配置平台”的高度。它为大规模、动态变化的数据采集需求提供了一个优雅且强大的解决方案，是实现真正意义上可持续、可维护的自动化的基石。

亮点五：多函数协同驱动的全流程闭环自动化工作流

`AutoDownloader` 不仅仅是实现了下载的自动化，更是通过多个设计精良、权责分明的函数协同工作，构建了一个从任务定义、环境准备、数据获取、执行下载到结果反馈的全流程闭环自动化工作流。其革命性体现在它将过去需要人工干预的多个步骤，无缝地整合到一个由代码驱动的、高效协作的“数字生产线”中。

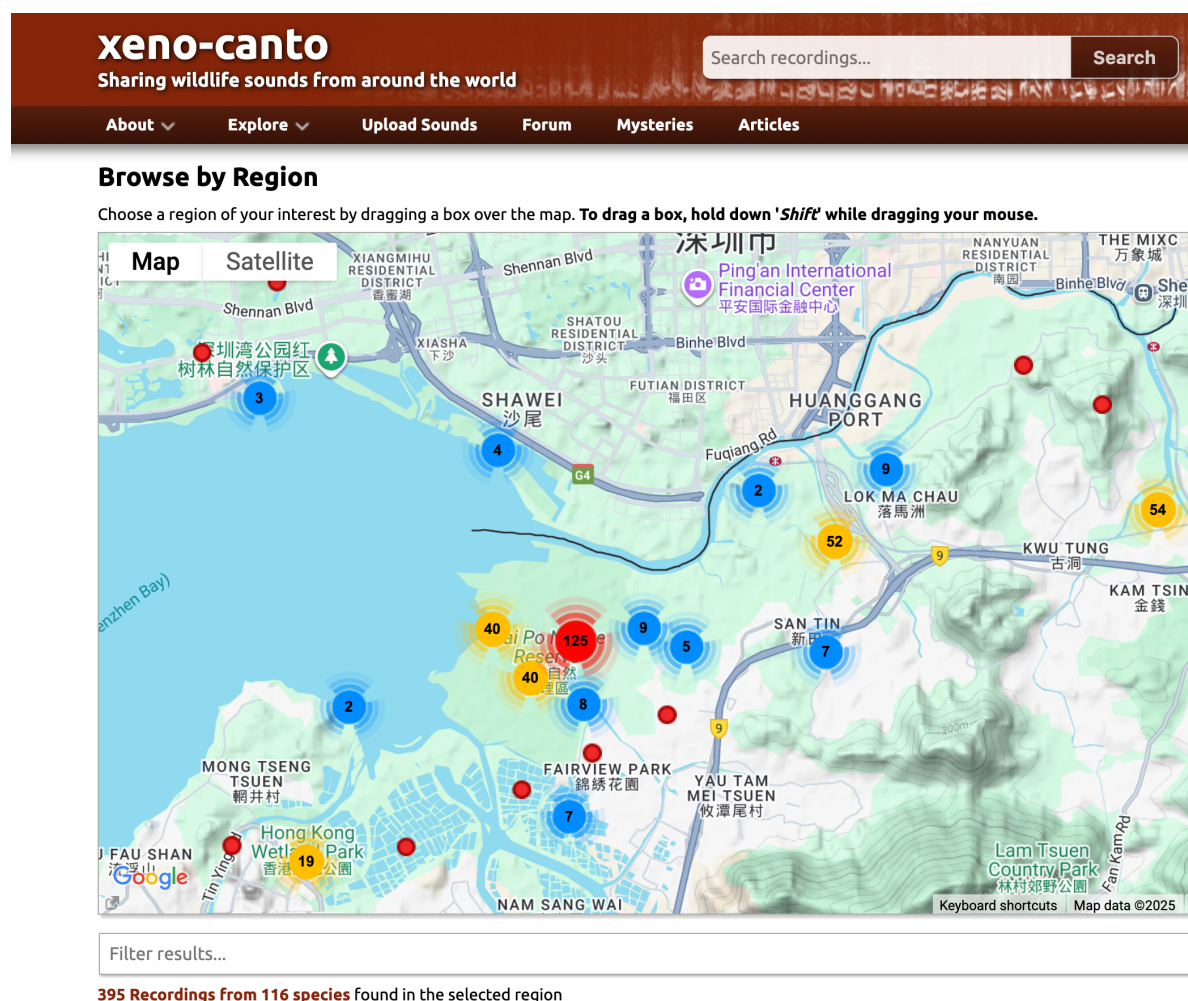
这条生产线的协同模式如下：

- **顶层入口 (`BatchDownloadXenoCanto`)：**作为顶层入口函数，它负责读取 CSV 任务清单，像一位项目总监一样，按顺序调度每一个物种的处理任务。它不关心具体如何下载，只负责发起任务、收集结果，并在最后生成 `download_log.csv` 总结报告。
- **子任务管理函数 (`DownloadXenoCantoSpecies`)：**这个函数负责管理单个物种的完整生命周期。接到总指挥的命令后，它首先调用“翻译官”`ToChineseName` 获取中文名，然后执行环境准备工作（如 `CreateDirectory` 创建规范化的文件夹）。接着，它会派遣“解析器”`UrlRetrieve` 去获取下载链接。拿到链接列表后，它亲自监督下载过程，记录成功与失败，并最终形成一份详细的 Association 格式的执行报告，返回给总指挥。
- **解析器 (`UrlRetrieve`)：**它的任务单一而明确：根据物种名，深入目标网站，智能地（按质量、按需）搜集回所有有效的下载链接。它将复杂的网络交互和页面解析逻辑封装起来，提供干净、直接的 URL 列表。
- **数据导入与调用 (`LoadSpeciesList & ToChineseName`)：**`LoadSpeciesList` 在任务开始时一次性加载所有物种的中英文对照表并缓存，构建了一个高效的内部“数据中心”。`ToChineseName` 则扮演了即时翻译的角色，在需要时提供快速查询服务。

这种清晰的函数协同机制，体现了现代软件工程的“单一职责原则”。每个函数都像一个专业的团队成员，只做自己最擅长的事，并通过定义清晰的输入（参数）和输出（返回值）与其他成员高效协作。这不仅使得代码极易阅读、理解和维护，更重要的是，它构建了一个坚固而灵活的自动化系统，其中任何一个环节（如解析逻辑）的修改都不会影响到其他部分，展现了工程设计水准。

2.3 BirdSoundRecognition_HK_SZ_Border 专题数据集

我们首先对 Xeno-Canto 数据库中所有标记为 米埔自然保护区 和保护区边界的录音进行统计，并选取了录音数量最多的 9 种鸟类作为目标。通过运行上述自动化爬虫，我们成功构建了名为 `BirdSoundRecognition_HK_SZ_Border` 的专题原始音频数据集，其总文件数为 466 个，总数据大小为 1.79 GB。这个经过初步筛选和整理的数据集，是后续所有特征工程步骤的原材料，其质量直接关系到最终特征的有效性。



第三章：基于 Mathematica 的鸟鸣信号处理与特征提取流程

一套完整的鸟鸣音频分析 workflow 始于原始音频的加载，通过高通滤波和标准化等一系列预处理操作来提纯信号。随后，利用时频谱（Spectrogram）、周期图（Periodogram）和局部振幅测量等分析工具，对鸟鸣在时域和频域上的核心特征进行分析与量化。此流程充分展示了 Mathematica 在生物声学研究中，从数据研究到数据清洗的强大一站式处理能力。

3.1. 频段分离与语音净化的理论基础

在城市环境中，声学背景主要受交通流、道路噪声及其他城市人造声源主导，其频谱能量集中在低频区段。实证研究发现，在城市交通噪声中，主要声能量集中于 2 kHz 以下。⁵此外，在有关城市-鸟类声学适应的研究中，发现鸟类在城市噪声背景下倾向于提高其鸣唱的最低频率以规避低频噪声掩蔽效应⁶。基于以上两条逻辑，一方面滤除低于 2 kHz 的频段，有效减少了城市交通噪声对鸟类鸣声的掩蔽干扰；另一方面，设定上限至 20 kHz（远高于常见鸟类鸣唱频段但仍处于可录制、分析的实际范围）则涵盖了绝大多数鸟类鸣唱频谱成分，从而

保证声学信号的完整性。综合来看，将滤波范围设定为 2 kHz 至 20 kHz，不仅有助于提升鸟类鸣唱信号的信噪比，而且在理论与实证研究的框架下具备良好的合理性。

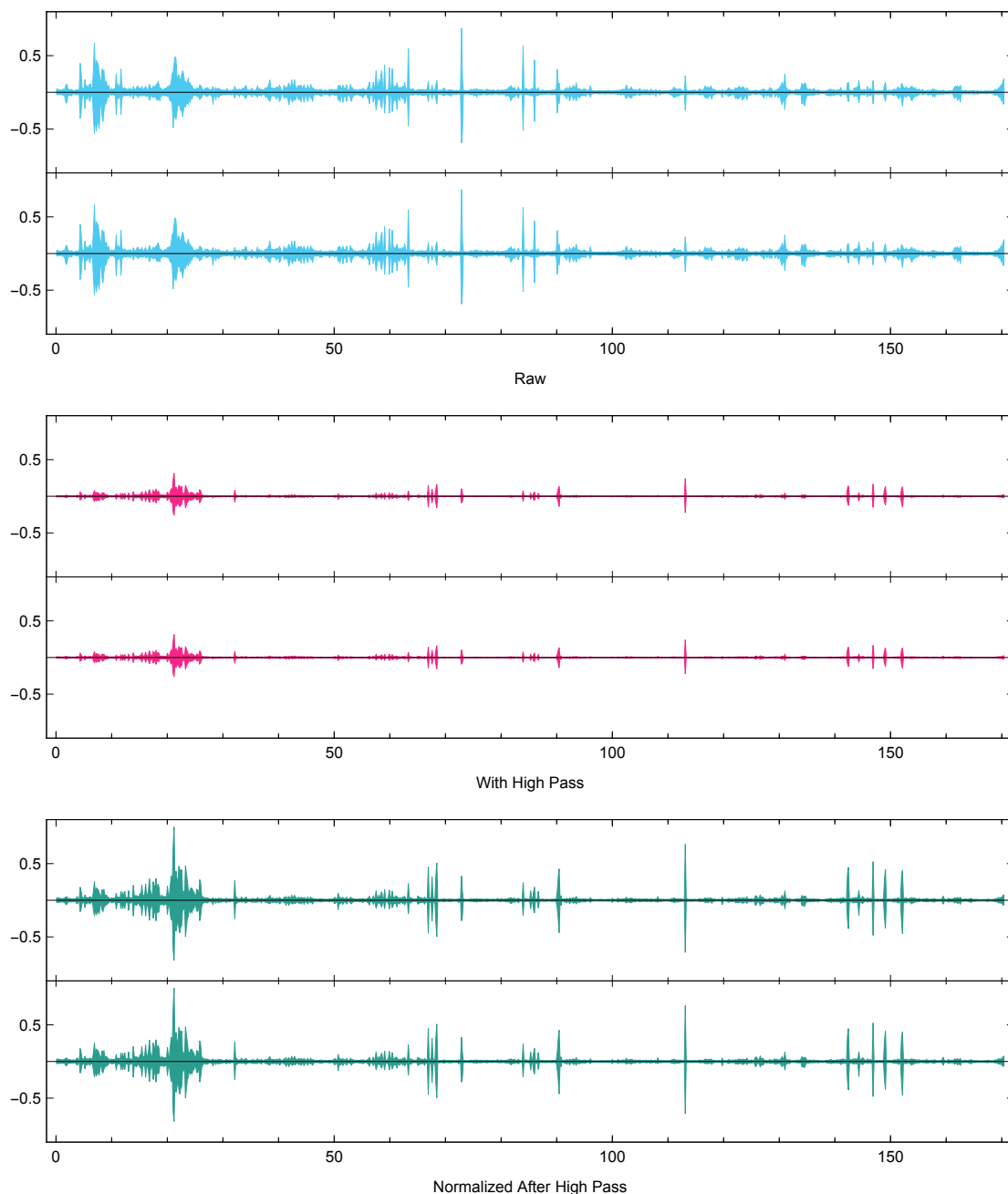
3.2. 数据加载与预处理流程的可视化对比

```
In[*]:= audio2 = audioDataset["834484"]
Column[{AudioPlot[audio2, FrameLabel → "Raw",
  PlotStyle → RGBColor["#4CC9F0"], ImageSize → Large],
  AudioPlot[HighpassFilter[audio2, Quantity[2000, "Hertz"]], FrameLabel →
    "With High Pass", PlotStyle → RGBColor["#F72585"], ImageSize → Large],
  AudioPlot[AudioNormalize[HighpassFilter[audio2, Quantity[2000, "Hertz"]]],
    FrameLabel → "Normalized After High Pass",
    PlotStyle → RGBColor["#2A9D8F"], ImageSize → Large]},
  Spacings → 1, BaseStyle → {FontSize → 16, FontWeight → "Bold"}
(*enlarge labels*)]
```

Out[*]=



Out[] =



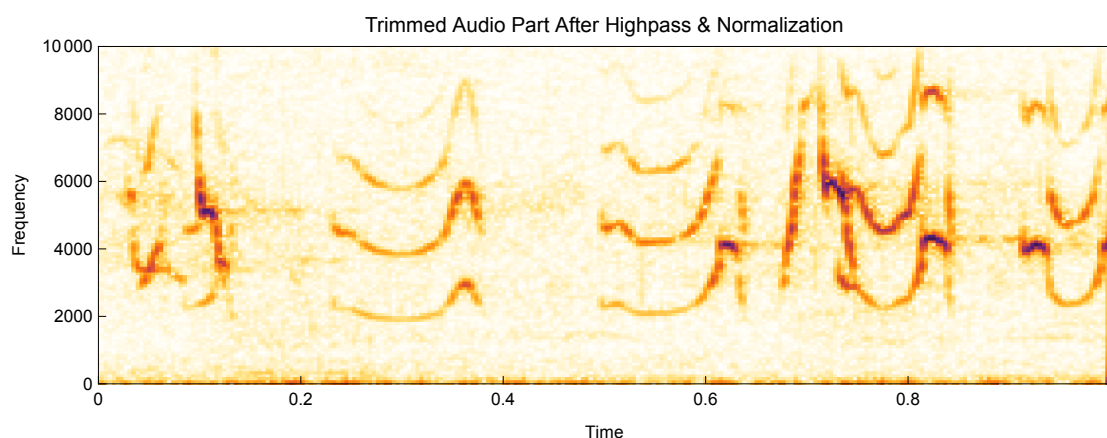
- `audio2 = audioDataset["834484"]`: 首先，加载ID为“834484”的音频样本，并将其存储在 `audio2` 变量中。
- (图1 - Raw): `AudioPlot` 函数将原始音频的双声道波形（振幅-时间图）绘制出来。这张天蓝色的图展示了未处理的信号，其中包含了目标鸟鸣以及可能存在的低频环境噪声（如风声）。
- (图2 - With High Pass): 接着，应用 `HighpassFilter` 函数，设定截止频率为 2000Hz。此操作移除了频率低于2000Hz的所有信号成分，旨在消除低频噪声。在第二幅洋红色的图中，我们可以看到信号的基线变得平坦，但整体振幅也因此减弱，使得鸟鸣的细节不突出。

- (图3 - Normalized After High Pass): 最后, `AudioNormalize` 函数被应用于已滤波的音频上。它将音频振幅等比例放大, 使其峰值达到最大。在第三幅青绿色的图中, 信号的动态范围得到了优化, 鸟鸣的精细结构——每一个音节的起伏和强度变化——都变得极为清晰, 为后续分析提供了高质量的数据。

3.3. 核心分析 (一): 时频谱 (Spectrogram) 分析

```
In[ ]:= Spectrogram[AudioNormalize[
  AudioTrim[HighpassFilter[audio2, Quantity[2000, "Hertz"]], {16, 17}]],
  ImageSize → Large, PlotLabel →
  "Trimmed Audio Part After Highpass & Normalization",
  FrameLabel → {"Time", "Frequency"}]
```

Out[]:=



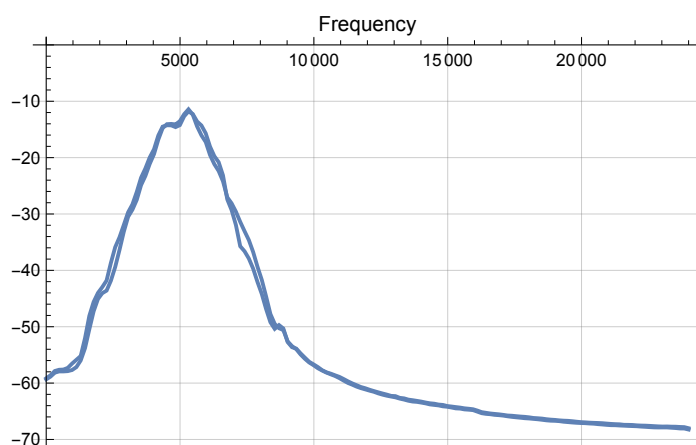
- 在生成时频谱之前, 使用 `AudioTrim` 函数精确截取了预处理后音频的第16秒到17秒的片段, 以聚焦于一段信息量丰富的鸣叫。
- `Spectrogram` 函数计算并绘制出该片段的时频谱。图像的横轴是时间, 纵轴是频率, 颜色的亮度代表能量强度。
- 这张图清晰地展示了鸟鸣的复杂频率调制模式。我们可以看到多条明亮、弯曲的能量轨迹, 这些是构成鸟鸣的基频和泛音。这些曲线的形状、斜率、间隔和持续时间都是识别物种、研究其行为和“语法”的关键特征。

3.4. 核心分析 (二)：如何通过对频域与时域特征的量化对音频进行精准裁切

3.4.A. 频域能量分布 (Periodogram)

```
In[*]:= Periodogram[
  AudioNormalize[HighpassFilter[audioDataset["720754"], Quantity[0, "Hertz"]]],
  300, 200, HammingWindow, GridLines -> Automatic,
  PlotLabel -> "Frequency", PlotLegends -> Automatic]
```

Out[*]=

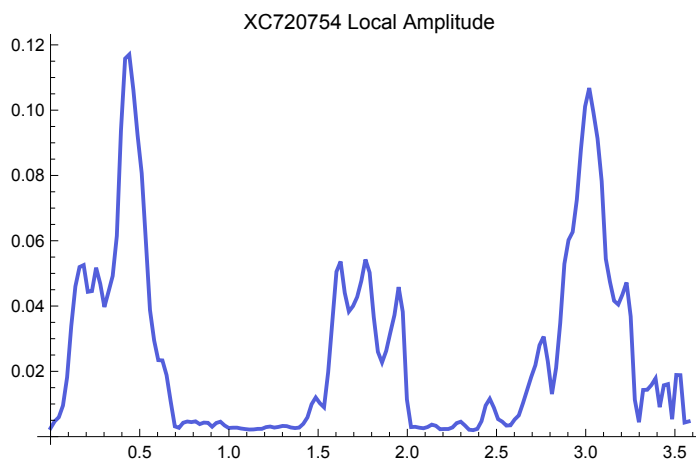


- **Periodogram** 函数（功率谱分析）计算了音频信号在整个频域上的能量分布。
- 其显示，该鸟鸣的能量存在一个非常明显的峰值，集中在 **5000 Hz** 左右。反映了我们主要声音信息的集中模式。

3.4.B. 分析时域能量包络 (Local Amplitude)

```
In[*]:= AudioLocalMeasurements[
  AudioNormalize[AudioTrim[HighpassFilter[audioDataset["720754"],
    Quantity[2000, "Hertz"]], {75.4, 79}]], "RMSAmplitude"];
ListLinePlot[%, PlotRange -> All,
  PlotLabel -> "XC720754 Local Amplitude", PlotStyle -> RGBColor["#535fdb"]]
```

Out[*]=



- `AudioLocalMeasurements`函数用于计算音频的局部均方根振幅，它能有效地描绘出声音能量随时间变化的包络。代码首先对音频 "XC720754" 进行了滤波和标准化，并截取了75.4秒至79秒的片段。
- 这幅图展示了鸟鸣的能量包络。图中的每一个波峰都对应着一个或一组鸣叫音节，而波谷则代表鸣叫的间歇。通过分析这条曲线的峰值、宽度和间隔，可以量化出鸟鸣的节律特征，例如音节率（syllable rate）和音节持续时间，这对于研究鸟类通讯的语法结构至关重要。
- 我们使用RMS进行窗口的确定与裁切。

确定函数的运用与封装

这部分催生了两个函数, 其中两个入口函数可输入XC - index为字符串或数字数据类型, 函数在内部使用`ToString[]`进行强制类型转换, 保证了稳健性（注：此处的`autoFindAndPlotMelSpectrogram`和`createAudioMatrixPlot`为未使用`safeMel224`的测试版本）【注意：该代码不包含数据导入模块，不可直接执行 / 执行代码请使用第四章的完整函数封装】

```
In[ ]:= ClearAll[MaxRMSAmplitudeTimeStamp];
MaxRMSAmplitudeTimeStamp[audioDataset_, keyIn_,
  sampleRate_Integer : 48 000, cutoff_ : Quantity[2000, "Hertz"],
  cutup_ : Quantity[10 000, "Hertz"]] := Module[{mea, maxVal, maxPos, len},
  key = ToString[keyIn];
  mea = AudioLocalMeasurements[
    AudioNormalize@BandpassFilter[Audio[audioDataset[key],
      SampleRate → sampleRate], {cutoff, cutup}], "RMSAmplitude"];
  maxVal = Max[mea];
  maxPos = FirstPosition[mea, maxVal];
  len = Length[mea["Path"]];
  N[Last[maxPos] / len * Duration[audioDataset[key]]]]

enc3 = NetEncoder[
  {"AudioMelSpectrogram", "SampleRate" → 48 000, (*<--在这里明确指定采样率*)
    "NumberOfFilters" → 224, "MinimumFrequency" → 2000,
    "MaximumFrequency" → 20 000, "WindowSize" → 1024, "TargetLength" → 224}];

ClearAll[createAudioMatrixPlot];
createAudioMatrixPlot[xcIndex_String,
  startTime_?NumericQ, endTime_?NumericQ] :=
MatrixPlot[Log[Transpose[enc3[AudioNormalize[AudioTrim[
  BandpassFilter[audioDataset[xcIndex], {Quantity[2000, "Hertz"],
    Quantity[20 000, "Hertz"]}], {startTime, endTime}]]]],
  DataReversed → {True, False}, FrameTicks → {{True, False}, {True, False}},
  FrameLabel → {"Frequency", "Time"},
  AspectRatio → 1, PlotLabel →
  StringJoin[{ToString[GetFolderInitial[xcIndex]], " - XC", xcIndex}],
  PlotLegends → Automatic]
```

```

ClearAll[autoFindAndPlotMelSpectrogram];
autoFindAndPlotMelSpectrogram[audioDataset_, key_] :=
Module[{keyStr, file, timeStampLocal, startTime, endTime},
  keyStr = ToString[key];
  If[! KeyExistsQ[audioDataset, keyStr],
    (*Try auto-import from base directory if file exists*)
    file = FileNameJoin[{baseDir, keyStr <> ".mp3"}];
    If[FileExistsQ[file],
      Print[Style["[Info] Auto-importing missing key \"" <> keyStr <> "\".",
        Darker[Green], Bold]];
      audioDataset[keyStr] = Import[file], (*If not found,
      print warning and stop*)Print[Style["[Warning] Audio \"" <>
        keyStr <> "\" not found in dataset or directory.", Red, Bold]];
      Return[$Failed];];];
  Quiet[
    Check[timeStampLocal = MaxRMSAmplitudeTimeStamp[audioDataset, keyStr];
      startTime =
        Max[Quantity[0, "Seconds"], timeStampLocal - Quantity[0.80, "Seconds"]];
      endTime = Min[timeStampLocal + Quantity[1.38, "Seconds"],
        Duration[audioDataset[keyStr]]];
      createAudioMatrixPlot[keyStr,
        QuantityMagnitude[startTime], QuantityMagnitude[endTime]], Print[Style[
        "[Error] Failed to process \"" <> keyStr <> "\".", Darker[Red], Bold]];
      $Failed]]]

In[ ]:= timeStamp = MaxRMSAmplitudeTimeStamp[audioDataset, "720754"]
timeStamp = MaxRMSAmplitudeTimeStamp[audioDataset, 803 155] (*兼容字符串/整数键*)

Out[ ]:=
1 min 7.6345 s

Out[ ]:=
0 min 16.2474 s

```

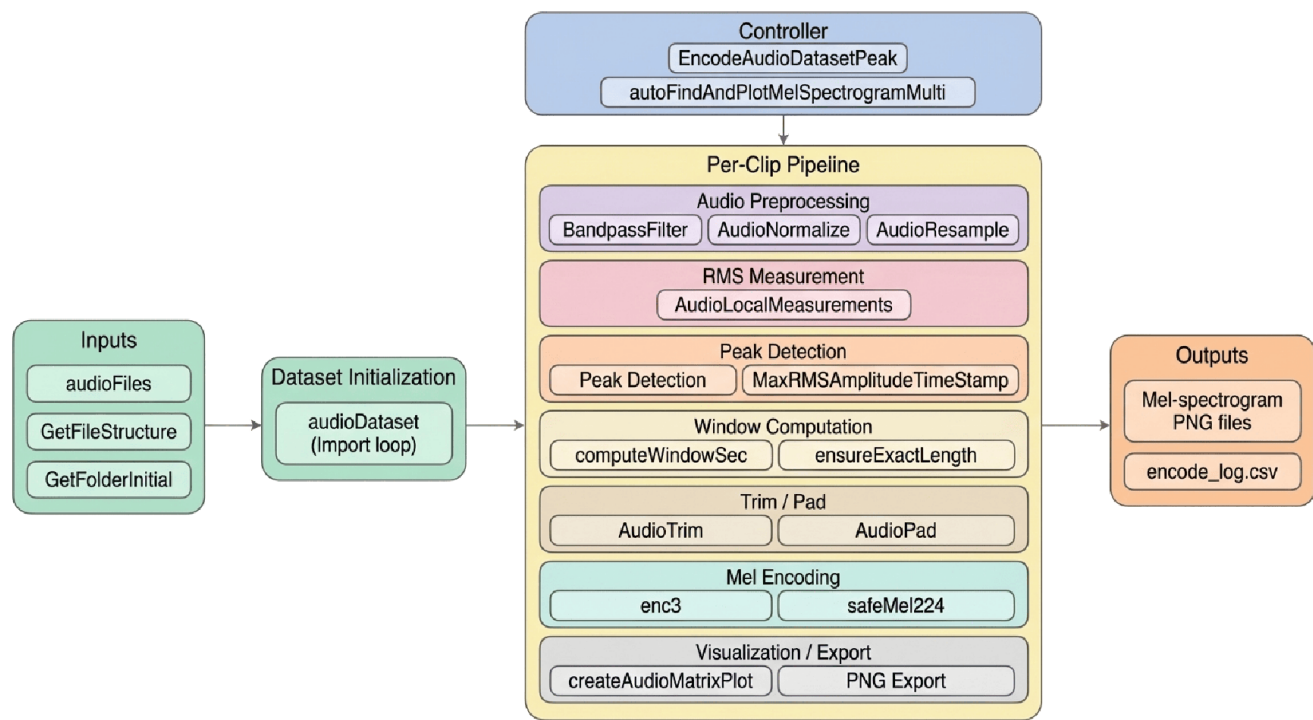
第四章：声学特征工程：从原始音频到Mel频谱指纹

在对原始音频处理之后，我们需要考虑特征提取的具体方式，并构建鲁棒的自动化流程函数。

4.1 为何选择梅尔频谱图？

为了将一维的时域音频信号转化为富含信息的频域二维图像化表示，我们选择了梅尔频谱图（Mel Spectrogram）。该特征在声学分析领域被广泛应用，其核心优势在于其频率轴（梅尔标度）并非线性，而是模拟了人类（及鸟类）听觉系统对频率的感知特性，即在低频分辨率更高，高频分辨率较低。这种仿生学的特性使得梅尔频谱图能够更有效地凸显鸟鸣声中的关键区分性特征⁷。

4.2 革命性的自动化特征提取流水线（完整函数封装）



Out[*]=

函数名 (Function Name)	核心作用 (Core Purpose)	输入 (Inputs)	选项 (Options)	返回值 (Return Value)
MaxRMSAmplitudeTimeStamp	定位能量峰值：计算并返回单个音频中RMS振幅最大点的时间戳。	audioDataset, key, sampleRate, cutoff, cutup	无	Quantity 对 (峰值时
safeMel224	安全生成梅尔谱：将 Audio 对象稳健地转换为 224x4x224 的梅尔频谱图矩阵，并处理所有数值异常。	aud (Audio 对象)	无	{matrix, {min, r (矩阵和范围)
createAudioMatrixPlot	可视化频谱图：根据指定的起止时间截取音频，并绘制其梅尔频谱图。	xcIndex, startTime, endTime	无	Graphics 对 (频谱图

autoFindAndPlotMelSpectrogram	单峰自动绘图：(简化版) 自动寻找能量最强的峰值，并绘制其周围窗口的频谱图。	audioDataset, key	无	Graphics 对象 \$Failed
computeWindowSec	精确计算窗口： 根据峰值时间、音频总长和窗长，智能计算出边界安全的截取起止点和所需填充时长。	tSec, durSec, wL, wR	无	{start, stop, padL, padR} (截取与填充参数)
ensureExactLength	修正音频长度： 通过微调裁剪或填充，确保一个 Audio 对象的时长严格等于目标值。	aud, Lsec	无	长度修正后的 Audio 对象
autoFindAndPlotMelSpectrogramMulti	单音频多峰处理：(核心执行) 对单个音频，寻找多个能量峰值，提取窗口，生成梅尔谱矩阵，并可选地导出 PNG 或绘图。	audioDataset, key	"NumWindows", "PeakMinDistance", "WindowBefore", "WindowAfter", "ExportPNG", "PlotPeaks", 等	List (包含多谱图或 \$Failed)
EncodeAudioDatasetPeaks	全数据集批量处理：(顶层调度) 遍历整个数据集，为每个音频调用多峰处理函数，并管理进度、日志和文件跳过逻辑。	audioDataset	继承自多峰函数的所有选项，外加 "SkipIfExists", "SaveLog", "LogFile"	Association (键为音频 ID，值为列表)

In[102]:=

```

(*=====START of INITIALIZATION=====*)
(*SetDirectory[
  "/Users/woody/Documents/UniDocs/BirdSoundRecognition_HK_SZ_Border"]; *)
baseDir = FileNameJoin[{$UserDocumentsDirectory, "XenoCantoDownloads"}];
SetDirectory[baseDir];
audioFiles = FileNames["*.mp3", baseDir, Infinity];
Short[audioFiles]
i = 0;

```

```

totalFiles = Length[audioFiles];
audioDataset = Monitor[Association[{i++;
    FileName[#] → Import[#]} & /@ audioFiles],
    ProgressIndicator[i / totalFiles, {0, 1}]];

ClearAll[GetFileStructure];
GetFileStructure[] := GetFileStructure[] = Module[{baseDir, audioFiles, rules},
    baseDir = FileNameJoin[{$UserDocumentsDirectory, "XenoCantoDownloads"}];
    audioFiles = FileNames["*.mp3", baseDir, Infinity];
    rules = Cases[audioFiles,
        path_ → Module[{fileName, folderIndex}, fileName = FileName[path];
            folderIndex = StringCases[FileName@DirectoryName[path],
                StartOfString ~~ digits : DigitCharacter .. ~~ __ → digits];
            If[folderIndex == {}, Nothing, fileName → First[folderIndex]]];
    Association[rules]];

ClearAll[GetFolderInitial];
GetFolderInitial[key_] :=
    Lookup[GetFileStructure[], ToString@key, Missing["NotFound"]];

(*=====END OF INITIALIZATION=====*)
(*=====START OF MODULE INITIALIZATION=====*)

ClearAll[MaxRMSAmplitudeTimeStamp];
MaxRMSAmplitudeTimeStamp[audioDataset_, keyIn_,
    sampleRate_Integer : 48 000, cutoff_ : Quantity[2000, "Hertz"],
    cutup_ : Quantity[10 000, "Hertz"]] := Module[{mea, maxVal, maxPos, len},
    key = ToString[keyIn];
    mea = AudioLocalMeasurements[
        AudioNormalize@BandpassFilter[Audio[audioDataset[key],
            SampleRate → sampleRate], {cutoff, cutup}], "RMSAmplitude"];
    maxVal = Max[mea];
    maxPos = FirstPosition[mea, maxVal];
    len = Length[mea["Path"]];
    N[Last[maxPos] / len * Duration[audioDataset[key]]]]

enc3 = NetEncoder[
    {"AudioMelSpectrogram", "SampleRate" → 48 000, (*<--在这里明确指定采样率*)
    "NumberOfFilters" → 224, "MinimumFrequency" → 2000,
    "MaximumFrequency" → 20 000, "WindowSize" → 1024, "TargetLength" → 224}];

safeMel224[aud_] := Module[{raw, m, finite, mn, mx}, raw = N@Transpose@enc3[aud];
    (*224×224, 非负*) m = N@Log[1. + raw];
    (*避免 Log[0]→-Inf*) (*清理任何非有限值*) m = Map[If[NumberQ[#] && FreeQ[#,
        Complex | Indeterminate | DirectedInfinity], N@#, 0.] &, m, {2}];
    m = Developer`ToPackedArray@m;
    finite = Select[Flatten[m],

```



```

    NumberQ[#] && FreeQ[#, Complex | Indeterminate | DirectedInfinity] &];
If[finite == {},
  {ConstantArray[0., {224, 224}], {0., 1.}}, {m, N@MinMax[finite]}]];

ClearAll[createAudioMatrixPlot];
createAudioMatrixPlot[xcIndex_String,
  startTime_?NumericQ, endTime_?NumericQ] :=
Module[{aud, mel, range}, aud = AudioNormalize@AudioResample[
  AudioTrim[BandpassFilter[audioDataset[xcIndex], {Quantity[2000, "Hertz"],
    Quantity[20000, "Hertz"]}], {startTime, endTime}],
  48000 (*与 enc3 的 SampleRate 对齐, 避免潜在不一致*)];
{mel, range} = safeMel224[aud];
(*取出矩阵与 {mn,mx}*)MatrixPlot[mel, (*不再 Log/Transpose*)
  DataReversed → {True, False}, FrameTicks → {{True, False}, {True, False}},
  FrameLabel → {"Frequency", "Time"}, AspectRatio → 1, PlotLabel →
  StringJoin[{ToString[GetFolderInitial[xcIndex]], " - XC", xcIndex}],
  PlotRange → range, (*复用安全范围*)PlotLegends → BarLegend[range]
  (*比 Automatic 更可控、更一致*)]]

ClearAll[autoFindAndPlotMelSpectrogram];
autoFindAndPlotMelSpectrogram[audioDataset_, key_] :=
Module[{keyStr, file, timeStampLocal, startTime, endTime},
  keyStr = ToString[key];
If[! KeyExistsQ[audioDataset, keyStr],
  (*Try auto-import from base directory if file exists*)
  file = FileNameJoin[{baseDir, keyStr <> ".mp3"}];
If[FileExistsQ[file],
  Print[Style["[Info] Auto-importing missing key \"" <> keyStr <> "\".",
    Darker[Green], Bold]];
  audioDataset[keyStr] = Import[file], (*If not found,
  print warning and stop*)Print[Style["[Warning] Audio \"" <>
    keyStr <> "\" not found in dataset or directory.", Red, Bold]];
  Return[$Failed];];];
Quiet[
  Check[timeStampLocal = MaxRMSAmplitudeTimeStamp[audioDataset, keyStr];
  startTime =
    Max[Quantity[0, "Seconds"], timeStampLocal - Quantity[0.80, "Seconds"]];
  endTime = Min[timeStampLocal + Quantity[1.38, "Seconds"],
    Duration[audioDataset[keyStr]]];
  createAudioMatrixPlot[keyStr,
    QuantityMagnitude[startTime], QuantityMagnitude[endTime]], Print[Style[
    "[Error] Failed to process \"" <> keyStr <> "\".", Darker[Red], Bold]];
  $Failed]]]

(* =====*)
(*安全 Mel 生成 (224x224) *)

```

```

(* =====*)
ClearAll[safeMel224];
safeMel224[aud_] :=
Module[{raw, m, finite, mn, mx}, raw = N@Transpose@enc3[aud];
  (*224x224, 非负*) m = N@Log[1. + raw];
  (*避免 Log[0]→-Inf*) (*清理任何非有限值*) m = Map[If[NumberQ[#] && FreeQ[#,
    Complex | Indeterminate | DirectedInfinity], N@#, 0.] &, m, {2}];
  m = Developer`ToPackedArray@m;
  finite = Select[Flatten[m],
    NumberQ[#] && FreeQ[#, Complex | Indeterminate | DirectedInfinity] &];
  If[finite == {},
    {ConstantArray[0., {224, 224}], {0., 1.}}, {m, N@MinMax[finite]}];

(* =====*)
(*计算“精确 2.18 s”截窗 (边界/补齐处理) *)
(*tSec:峰值秒;durSec:音频总秒;wL/wR:左右窗秒*)
(*返回: {startSec,stopSec,padLeftSec,padRightSec}*)
(* =====*)
(*Robust:keep exact length L=wL+wR;prefer centering at tSec;
shift left/right if touching boundaries;pad only if dur<L.*)
ClearAll[computeWindowSec];
computeWindowSec[tSec_, durSec_, wL_, wR_] :=
Module[{L = N[wL + wR], start, stop, padL = 0., padR = 0., shift, eps = 1.*^-9},
  If[durSec + eps < L, (*Too short:take full clip,
    pad to L (split per desired offsets)*) start = 0.;
    stop = durSec;
    padL = Max[0., wL - tSec];
    padR = Max[0., wR - (durSec - tSec)];
    If[padL + padR < L - (stop - start) - eps,
      padR = padR + (L - (stop - start) - (padL + padR))];
    Return[{start, stop, padL, padR}];];
  (*Ideal window centered at tSec*) start = tSec - wL;
  stop = tSec + wR;
  (*Shift right if left underflows*) If[start < -eps, shift = -start;
    start += shift;
    stop += shift];
  (*Shift left if right overflows*)
  If[stop > durSec + eps, shift = stop - durSec;
    start -= shift;
    stop -= shift];
  (*Final clamp to be safe and enforce exact L*)
  start = Clip[start, {0., durSec - L}];
  stop = start + L;
  {start, stop, 0., 0.}];

(*After trimming/padding,
force exact L seconds (handles quantization drift).*)

```

```

ClearAll[ensureExactLength];
ensureExactLength[aud_, Lsec_] :=
Module[{len = QuantityMagnitude[Duration[aud], "Seconds"], eps = 1.*^-6, need},
Which[Abs[len - Lsec] ≤ eps, aud, len < Lsec - eps,
AudioPad[aud, {0, Quantity[Lsec - len, "Seconds"]}], True,
AudioTrim[aud, {Quantity[0, "Seconds"], Quantity[Lsec, "Seconds"]}]];

(* =====*)
(*仅峰值模式：多窗 Mel 生成 (含可视化、精确 2.18s、safe 导出) *)
(* =====*)
ClearAll[autoFindAndPlotMelSpectrogramMulti];

Options[autoFindAndPlotMelSpectrogramMulti] =
{
"NumWindows" → 5, (*取几个峰对应的窗*) "PeakMinDistance" → 0.3,
(*峰间最小时间间隔 (秒)*) "WindowBefore" → 0.8, (*窗左侧时长 (秒)*)
"WindowAfter" → 1.38, (*窗右侧时长 (秒)*) "SampleRate" → 48 000,
"BandLow" → 2000, "BandHigh" → 20 000, "ExportPNG" → True,
(*是否导出 PNG*) "PlotPeaks" → True, (*是否绘 RMS+窗带*)
"OutputDir" → "mel_png_224_peak" (*相对 baseDir 的子目录*);
}

autoFindAndPlotMelSpectrogramMulti[audioDataset_, key_, OptionsPattern[]] :=
Module[{(*入参与选项*) keyAny, keyStr = ToString[key],
numW = OptionValue["NumWindows"], minDist = OptionValue["PeakMinDistance"],
wL = OptionValue["WindowBefore"], wR = OptionValue["WindowAfter"],
sr = OptionValue["SampleRate"], fL = OptionValue["BandLow"],
fH = OptionValue["BandHigh"], exportQ = OptionValue["ExportPNG"],
plotQ = OptionValue["PlotPeaks"], outSub = OptionValue["OutputDir"],
outDir, (*音频/测量*) aud, mea, vals, len, durSec, times, (*峰值检测*)
idxs, peakIdxAll, peakPairs, sortedIdx, pickedIdx = {}, minIdxDist, tSecs,
(*可视化*) maxVal, rects, vlines, rmsPlot, (*输出与导出*) mats = {}, i,
startSec, stopSec, padL, padR, audTrim, melPack, mm, mn, mx, img, baseName},
(*兼容字符串/整数键*) keyAny = Which[KeyExistsQ[audioDataset, key],
key, KeyExistsQ[audioDataset, keyStr], keyStr, True,
Print[Style["[Warning] Audio key not found: "<>keyStr, Red, Bold]]];
Return[$Failed]];

(*输出目录：若需要创建则打印一行信息；已存在则静默*)
(*outDir=FileNameJoin[{baseDir,outSub}]]);*)

species = ToString@GetFolderInitial[keyStr];
If[species === "Missing[NotFound]" || species === "", species = "Unknown"];

outDir = FileNameJoin[{baseDir, outSub, species}];

If[exportQ && ! DirectoryQ[outDir],
Quiet@CreateDirectory[outDir, CreateIntermediateDirectories → True];

```

```

Print[Style["[Info] Created folder: "<>outDir, Darker[Green], Bold]]];
(*统一预处理：带通+归一化+采样率*)
aud = AudioNormalize@BandpassFilter[Audio[audioDataset[keyAny],
  SampleRate → sr], {Quantity[fL, "Hertz"], Quantity[fH, "Hertz"]}];
(*RMS 曲线*)mea = AudioLocalMeasurements[aud, "RMSAmplitude"];
vals = N@Normal@mea["Values"]; (*1D 数组*)len = Length[vals];
If[len < 3,
  Print[Style["[Warning] Signal too short: "<>keyStr, Orange, Bold]];
  Return[$Failed]];
durSec = QuantityMagnitude[Duration[aud], "Seconds"];
times = Subdivide[0., durSec, len - 1];
(*手写局部峰值：大于左邻且不小于右邻*)idxs = Range[2, len - 1];
peakIdxAll = Select[idxs, vals[[#]] > vals[[# - 1]] && vals[[#]] ≥ vals[[# + 1]] &];
If[peakIdxAll === {}, (*兜底：没有局部峰，用全局最大*)
  tSecs = {(Ordering[vals, -1][[1]] - 1) * (durSec / (len - 1))}, (*有局部峰：
    按幅值降序+最小时间间隔筛选*)peakPairs = ({#, vals[[#]]} & /@ peakIdxAll);
  (*{idx, amp}*)sortedIdx = First /@ Reverse@SortBy[peakPairs, Last];
  (*idx 按 amp 降序*)minIdxDist = Max[1, Round[minDist / (durSec / (len - 1))]];
  (*秒→索引间隔*)pickedIdx = {};
  Do[If[Length[pickedIdx] < numW &&
    (pickedIdx === {} || Min[Abs[idx - pickedIdx]] ≥ minIdxDist),
    AppendTo[pickedIdx, idx]], {idx, sortedIdx}];
  If[pickedIdx === {},
    tSecs = {(Ordering[vals, -1][[1]] - 1) * (durSec / (len - 1))},
    tSecs = (pickedIdx - 1) * (durSec / (len - 1))];];
(*可视化：RMS 曲线+灰色窗带+红色峰线*)
If[plotQ, maxVal = Max[vals];
  (*rects=Table[{GrayLevel[0.9],Opacity[0.35],
    Rectangle[{Max[0.,t-wL],0},{Min[durSec,t+wR],maxVal}]}],{t,tSecs}];*)
  rects = Table[
    Module[{s, e, pl, pr}, {s, e, pl, pr} = computeWindowSec[t, durSec, wL, wR];
    {GrayLevel[0.9], Opacity[0.35], Rectangle[{s, 0}, {e, maxVal}]}],
    {t, tSecs}];

  vlines = Table[{Red, Dashed, Line[{t, 0}, {t, maxVal}]}], {t, tSecs}];
  rmsPlot =
    Show[ListLinePlot[Transpose[{times, vals}], Frame → True, PlotRange → All,
      ImageSize → 520, FrameLabel → {"时间 (s)", "RMS 幅值"}, PlotLabel →
        "RMS 与截窗 (峰值模式) "<>keyStr], Graphics[rects], Graphics[vlines]];
  Print[rmsPlot];];
(*逐峰生成“精确 2.18s”窗→Mel→安全导出*)
Do[{startSec, stopSec, padL, padR} =
  computeWindowSec[tSecs[[i]], durSec, wL, wR];
  audTrim = AudioNormalize@AudioTrim[aud,
    {Quantity[startSec, "Seconds"], Quantity[stopSec, "Seconds"]}];
  If[padL > 0. || padR > 0., audTrim = AudioPad[audTrim,

```

```

    {Quantity[padL, "Seconds"], Quantity[padR, "Seconds"]}]];];
{melPack, mm} = safeMel224[audioTrim];
(*224x224 有限矩阵*) {mn, mx} = mm;
img = If[mx == mn, Image[ConstantArray[0.5, {224, 224}], "Real"],
  (*常数兜底中灰*) Image[Rescale[melPack, {mn, mx}], "Real"]];
baseName = keyStr <> "_peak" <> ToString[i] <> ".png";
If[exportQ, Export[FileNameJoin[{outDir, baseName}], img]];
AppendTo[mats, melPack];, {i, Min[numW, Length[tSecs]]}];
Print[Style["    Generated " <> ToString[Length[mats]] <>
  " peak windows for XC" <> keyStr, Darker[Green], Bold]];
mats];

ClearAll[EncodeAudioDatasetPeak];
Options[EncodeAudioDatasetPeak] =
{"NumWindows" → 5, (*每个音频截取的峰值窗个数*) "PeakMinDistance" → 0.3,
  (*峰间最小时间间隔(秒)*) "WindowBefore" → 0.8, (*窗左侧时长(秒)*)
  "WindowAfter" → 1.38, (*窗右侧时长(秒)*) "SampleRate" → 48000,
  "BandLow" → 2000, "BandHigh" → 20000, "OutputSubfolder" → "mel_png_224_peak",
  (*相对 baseDir 的子目录*) "ExportPNG" → True, (*是否导出 PNG*)
  "PlotPeaks" → False, (*批量跑默认不画图*) "SkipIfExists" → True,
  (*若该 key 的目标 PNG 已存在则跳过*) "SaveLog" → True, (*导出日志 CSV*)
  "LogFile" → "encode_log.csv" (*日志文件名(写到 baseDir 下)*)};

EncodeAudioDatasetPeak[audioDataset_, opts : OptionsPattern[]] :=
Module[{keys, total, i = 0, numW = OptionValue["NumWindows"],
  outSub = OptionValue["OutputSubfolder"], outDir,
  exportQ = OptionValue["ExportPNG"], plotQ = OptionValue["PlotPeaks"],
  skipQ = OptionValue["SkipIfExists"], saveLogQ = OptionValue["SaveLog"],
  logFile = OptionValue["LogFile"], minDist = OptionValue["PeakMinDistance"],
  wL = OptionValue["WindowBefore"], wR = OptionValue["WindowAfter"],
  sr = OptionValue["SampleRate"], fL = OptionValue["BandLow"],
  fH = OptionValue["BandHigh"], results = <|>, logRows = {}, mats, keyStr,
  pngs, allExistQ, t0 = DateString[], keys = Keys[audioDataset];
total = Length[keys];
(*1) 输出目录: 若不存在则创建并打印 info; 若存在则静默*)
outDir = FileNameJoin[{baseDir, outSub}];
If[exportQ && ! DirectoryQ[outDir],
  Quiet@CreateDirectory[outDir, CreateIntermediateDirectories → True];
  Print[Style["[Info] Created folder: " <> outDir, Darker[Green], Bold]]];];
(*2) 遍历编码*)
Monitor[Do[i++; keyStr = ToString[keys[[k]]];
  (*若设置了跳过已存在, 则检测 key 对应的 PNG 是否齐全*) allExistQ = False;
  If[exportQ && skipQ, pngs = Table[FileNameJoin[
    {outDir, keyStr <> "_peak" <> ToString[j] <> ".png"}], {j, numW}];
    allExistQ = And@@ (FileExistsQ /@ pngs)];];
  If[allExistQ, (*记录并跳过*)

```

```

AppendTo[logRows, {keyStr, 0, "skipped_exists", outDir, DateString[]}];
Continue[]];];
mats = Check[autoFindAndPlotMelSpectrogramMulti[audioDataset,
  keyStr, "NumWindows" → numW, "PeakMinDistance" → minDist,
  "WindowBefore" → wL, "WindowAfter" → wR, "SampleRate" → sr,
  "BandLow" → fL, "BandHigh" → fH, "ExportPNG" → exportQ,
  "PlotPeaks" → plotQ, "OutputDir" → outSub], $Failed];
If[mats === $Failed,
  AppendTo[logRows, {keyStr, 0, "failed", outDir, DateString[]}];
  results[keyStr] = $Failed,
  AppendTo[logRows, {keyStr, Length[mats], "ok", outDir, DateString[]}];
  results[keyStr] = mats];, {k, 1, total}], Row[{"Encoding ", i, "/",
  total, "  (" , NumberForm[100.0 i / Max[1, total], {3, 1}], "%)" ]];
(*3) 导出日志*)
If[saveLogQ, With[{logPath = FileNameJoin[{baseDir, logFile]}],
  Export[logPath, Prepend[logRows,
    {"Key", "Windows", "Status", "OutputDir", "Timestamp"}], "CSV"];
  Print[Style["[Info] Saved log: " <> logPath, Darker[Blue], Bold]]];];];
Print[Style["[Done] Encoded keys: " <>
  ToString[Count[Values[results], _List]] <> " / " <> ToString[total] <>
  "  (" <> t0 <> " → " <> DateString[] <> ")", Darker[Green], Bold]];
results];

(*=====END OF MODULE INITIALIZATION=====*)

```

```

Out[103]=
/Users/woody/Documents/XenoCantoDownloads; Null

Out[104]//Short=
{ /Users/woody/Documents/XenoCantoDownloads/1
  Plain_Prinia (褐头鹪莺)/1000797.mp3,
  <<414>>, /Users/woody/Documents/XenoCantoDownloads/9
  White-breasted_Waterhen (白腹秧鸡)/996617.mp3}

```

4.3. AutoEncoder 的八个核心亮点

亮点一. 事件驱动的智能信号聚焦 (MaxRMSAmplitudeTimeStamp & 峰值检测)

我们编写了一个峰值检测模块，进一步升级拓展为可控的多峰值检测信号聚焦

```

In[*]:= timeStamp = MaxRMSAmplitudeTimeStamp[audioDataset, "720754"]
timeStamp = MaxRMSAmplitudeTimeStamp[audioDataset, "803155"]

Out[*]=
1 min 7.6345 s

Out[*]=
0 min 16.2474 s

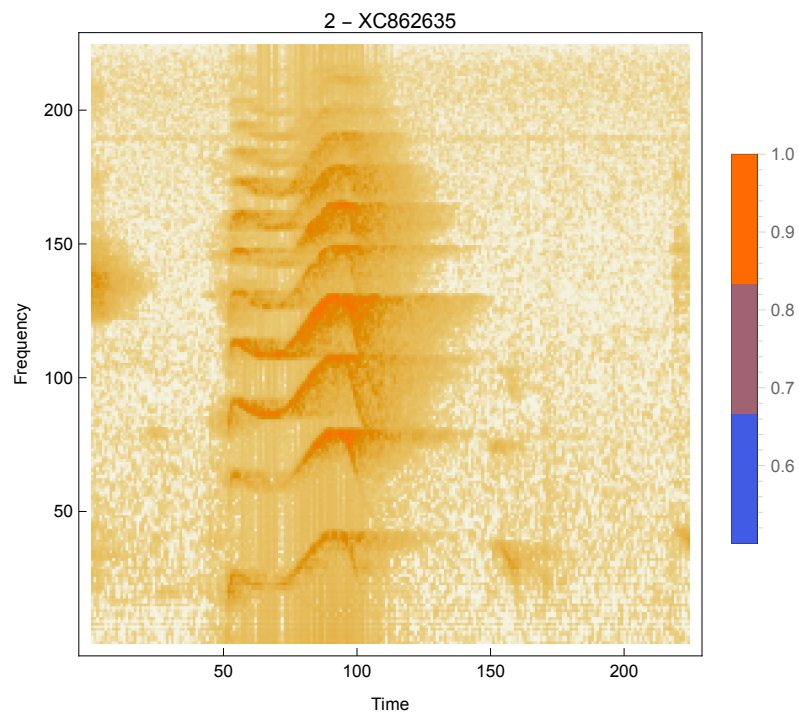
```


初始化后，只需通过调用函数`autoFindAndPlotMelSpectrogram`，就可以画出任意录音第一个峰值信号的Mel频谱图作为该录音的声学指纹

```
In[132]:=
```

```
autoFindAndPlotMelSpectrogram[audioDataset, 862 635]
```

```
Out[132]=
```



利用`AudioTrim`和`MaxRMSAmplitudeTimeStamp`，我们可以轻易地得到该切片的音频预览：

```
In[ ]:= audio1 = audioDataset["862635"];
```

```
ts = MaxRMSAmplitudeTimeStamp[audioDataset, 862 635];
```

```
AudioNormalize@
```

```
BandpassFilter[#, {Quantity[2000, "Hertz"], Quantity[20 000, "Hertz"]}] &@
```

```
AudioTrim[audio1, {ts - Quantity[0.8, "Seconds"], ts + Quantity[1.2, "Seconds"]}]
```

```
AudioPlot[%]
```

```
ListLinePlot[
```

```
AudioLocalMeasurements[#, "RMSAmplitude"] &@AudioNormalize@BandpassFilter[#,
```

```
{Quantity[2000, "Hertz"], Quantity[20 000, "Hertz"]}] &@AudioTrim[
```

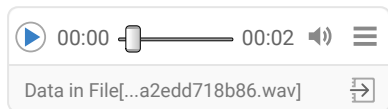
```
audio1, {ts - Quantity[0.8, "Seconds"], ts + Quantity[1.2, "Seconds"]}],
```

```
PlotRange → All, PlotLabel → "XC862635 Local Amplitude",
```

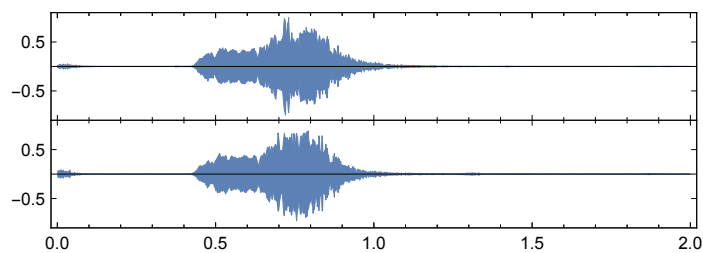
```
PlotStyle → RGBColor["#535fdb"],
```

```
AxesLabel → {Time, Amplitude}]
```

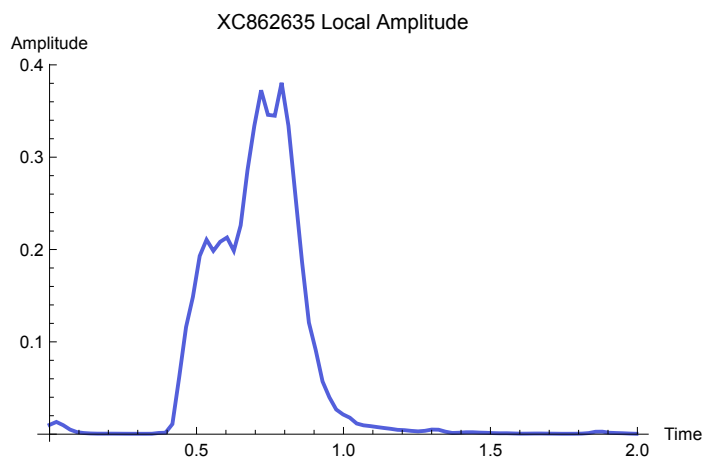
```
Out[ ]:=
```



```
Out[ ]:=
```



```
Out[ ]:=
```



进一步拓展为多峰值检测模块，其摒弃了传统音频处理中盲目、暴力的滑动窗口或随机切片方法，采用了一种基于信号内容的事件驱动策略，这在自动化处理中是革命性的一步。其核心思想是：在经过噪声滤波后，有价值的信息（如一声鸟鸣）必然伴随着能量的显著变化。代码首先通过 `AudioLocalMeasurements[aud, "RMSAmplitude"]` 计算出反映能量的RMS振幅曲线，然后通过一个精巧的自定义峰值检测算法 `Select[idxs, vals[[#]] > vals[[# - 1]] && vals[[#]] >= vals[[# + 1]] &]`，精确地定位出所有能量的局部峰值点。这一步的创造性在于，它将分析的焦点从时间转移到了事件。系统不再是漫无目的地扫描整段音频，而是

像一位声学专家一样，自动地将注意力集中在那些最可能包含关键发声事件的时刻。这种方法极大地提升了数据信噪比，确保了后续提取的特征都源自高信息量的音频片段，从源头上为深度学习模型的性能提供了坚实保障。它将原本需要人工标记的数据清洗环节完全自动化、精准化，为处理海量、未经清洗的野外录音数据提供了前所未有的强大工具。

亮点二：数值安全的健壮特征生成 (safeMel224)

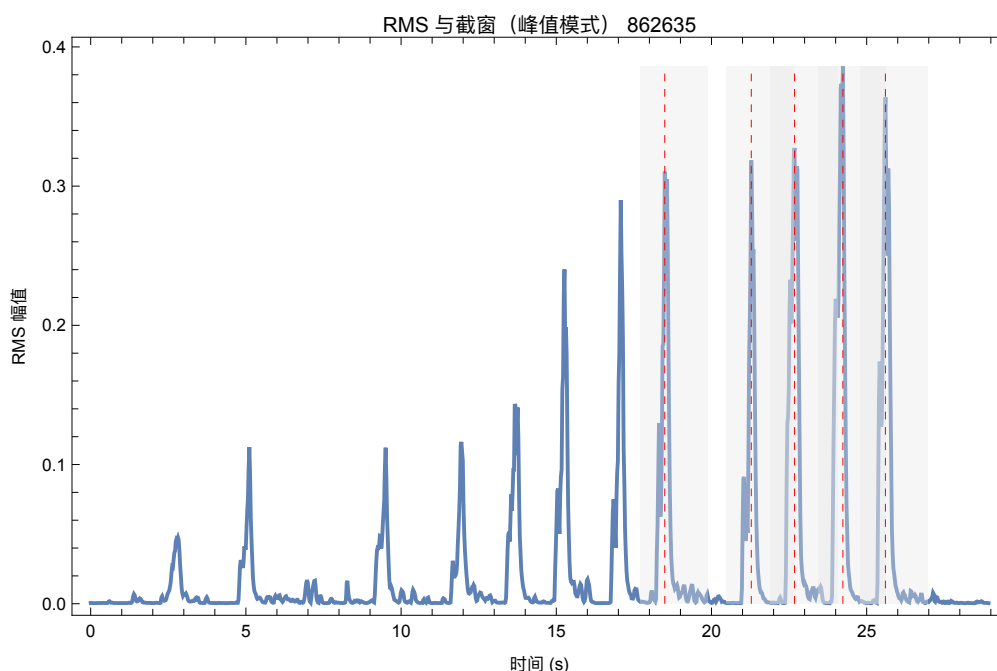
在将音频片段转化为梅尔频谱图的过程中，该系统展现了严谨性与鲁棒性。函数 `safeMel224` 是整个系统稳定运行的守护神，它预见并处理了所有可能导致计算崩溃或产生无效数据（NaN, Infinity）的陷阱。首先，它使用 `N@Log[1. + raw]` 而非简单的 `Log[raw]` 来进行对数缩放。这个微小 `+ 1` 的改动，创造性地解决了当频谱能量 `raw` 为零时 `Log[0]` 会产生 `-Infinity` 的致命问题，这是一个简单却极其有效的数值稳定技巧。其次，代码通过 `Map[If[NumberQ[#] && FreeQ[#, Complex | Indeterminate | DirectedInfinity], N@#, 0.] &, m, {2}]` 对生成的矩阵进行了一次彻底的净化。这个模式匹配检查确保了矩阵中的每一个元素都是一个干净的实数，任何因上游计算意外产生的复数、不确定值或无穷大，都会被安全地替换为零，从而避免了将脏数据喂给神经网络。最后，`DeveloperToPackedArray@m` 将处理好的数据转换为性能更高的打包数组。`safeMel224` 的存在，意味着无论上游输入何种质量的音频，它都能保证输出一个格式统一、数值安全、计算高效的特征矩阵，这种绝对可靠的自动化数据转换能力，是构建任何稳定深度学习系统的基石。

亮点三：革命性的边界感知精密切窗技术 (computeWindowSec)

该流水线最核心的创新之一，在于其彻底解决了音频切片中长期存在的边界效应与长度不一致问题，实现了革命性的自动化精密切窗。

```
In[ ]:= autoFindAndPlotMelSpectrogramMulti[audioDataset, "862635"];
```

```
[Info] Created folder: /Users/woody/Documents/XenoCantoDownloads/mel_png_224_peak/2
```



```
Generated 5 peak windows for XC862635
```

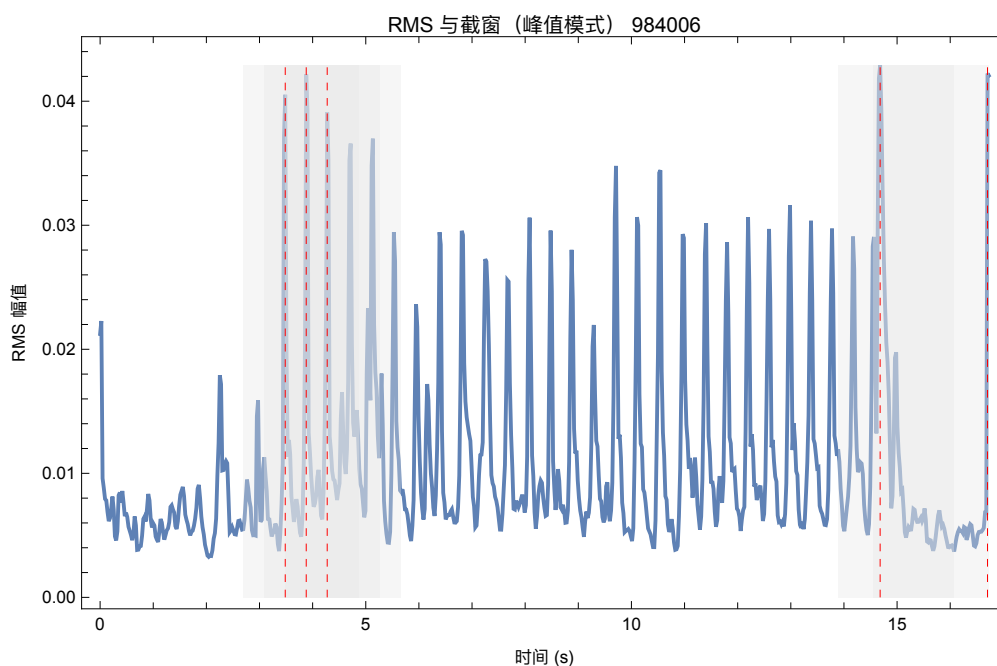
函数 `computeWindowSec` 的设计堪称完美，它能应对所有极端情况。当神经网络要求输入固定长度（例如2.18秒）的音频片段时，该函数首先通过 `If[durSec + eps < L, ...]` 判断原始

音频是否过短。如果是，它不会简单地报错或丢弃，而是智能地计算出 `padL` 和 `padR`，即左右两边需要填充多少静音，以凑成标准长度，保证了宝贵数据的最大化利用。更具革命性的是其边界处理逻辑：当理想窗口 `start = tSec - wL`; `stop = tSec + wR`; 超出音频的开头或结尾时，它并非粗暴地裁剪掉超出部分（这会导致信息丢失和长度变化），而是通过 `shift = -start`; `start += shift`; `stop += shift` 这样的代码，将整个窗口进行平移，在保证窗口总长度不变的前提下，使其整体滑回音频的有效范围内。这种保长平移策略，确保了每一个送入编码器的音频片段在物理时长上都是绝对统一的，这对于模型训练的稳定性和最终性能至关重要。这个函数是系统鲁棒性的终极体现，它将数据预处理中最为繁琐和易错的环节，变成了一个完全自动化、无需人工干预的精确操作。

右边界情况如下，我们的切片函数可以检测到异常并进行自动进行滑动窗口处理，同时Plot函数也可以正确绘制选取的区间并可视化地显示在全局RMS图中：

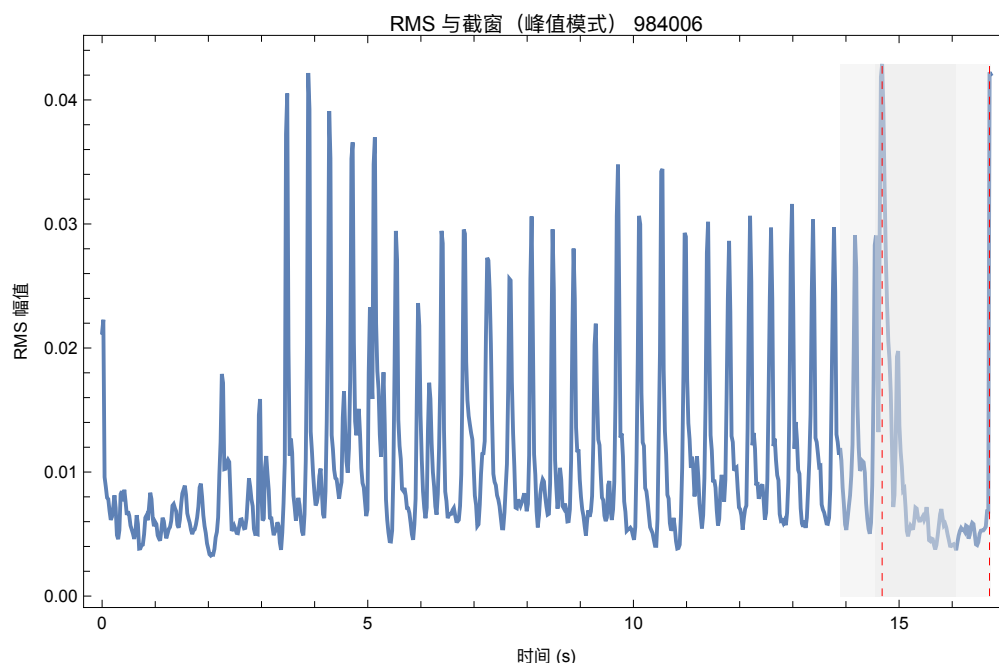
```
In[ ]:= autoFindAndPlotMelSpectrogramMulti[audioDataset, "984006",
  "NumWindows" → 5, "PeakMinDistance" → 0.3,
  "PlotPeaks" → True, "ExportPNG" → True,
  "OutputDir" → "mel_png_224_peak_test_ingte"
];
```

[Info] Created folder:
/Users/woody/Documents/XenoCantoDownloads/mel_png_224_peak_test_ingte/9



Generated 5 peak windows for XC984006

```
In[ ]:= autoFindAndPlotMelSpectrogramMulti[audioDataset, "984006",
  "NumWindows" → 2, "PeakMinDistance" → 0.3,
  "PlotPeaks" → True, "ExportPNG" → True,
  "OutputDir" → "mel_png_224_peak_test_ingte"
];
```



Generated 2 peak windows for XC984006

亮点四：多事件提取与时域去噪 (autoFindAndPlotMelSpectrogramMulti)

我们并未满足于仅仅找到能量最强的那个点，而是进化到了能够自动提取和筛选多个关键事件的更高维度。

在 `autoFindAndPlotMelSpectrogramMulti` 函数中，系统不仅找到了所有的能量峰值，还引入了两个关键的筛选机制，这在自动化分析中具有革命性意义。首先，通过 `sortedIdx = First /@ Reverse@SortBy[peakPairs, Last]`，它按照能量从高到低对所有峰值进行排序，确保了优先分析最重要的事件。其次，也是最关键的，它引入了时域去噪（或称事件分离）的逻辑：

`Min[Abs[idx - pickedIdx]] >= minIdxDist`。这里的 `minIdxDist`（由用户设置的“PeakMinDistance”选项换算而来）规定了两个被选中的事件之间必须保持的最小时间距离。这个机制极具创造性，它能有效避免对同一次发声（例如一声鸟鸣的多个能量起伏）进行重复采样，确保提取出的每个窗口都对应一个相对独立的声学事件。这相当于在时域上对事件进行了非极大值抑制，极大地提高了样本的多样性。

这种从单点分析到多点、去冗余分析的进化，使得系统能够从一段录音中自动挖掘出多个音频事件点，而不仅仅是一个孤立的片段，极大地提高了单一音频的复用率。

亮点五：高度可配置的模块化实验平台 (OptionsPattern)

该流水线通过广泛而深入地使用 Mathematica 的 `OptionsPattern` 机制，将自身从一个固定的脚本，升级为一个高度灵活、可配置的科学实验平台。在 `autoFindAndPlotMelSpectrogramMulti` 和 `EncodeAudioDatasetPeak` 这两个核心函数中，几乎所有的关键参数，从信号处理层面的采样率（“SampleRate”）、带通频率（“BandLow”，“BandHigh”），到事件提取层面的窗口数量（“NumWindows”）、峰间距（“PeakMinDistance”），再到 workflow 层面的文件导出开关（“ExportPNG”）和输出目录（“OutputDir”），都被设计成了可选项。

```

In[*]:= Options[autoFindAndPlotMelSpectrogramMulti]
Out[*]=
{NumWindows → 5, PeakMinDistance → 0.3, WindowBefore → 0.8,
 WindowAfter → 1.38, SampleRate → 48 000, BandLow → 2000, BandHigh → 20 000,
 ExportPNG → True, PlotPeaks → True, OutputDir → mel_png_224_peak}

In[*]:= Options[EncodeAudioDatasetPeak]
Out[*]=
{NumWindows → 5, PeakMinDistance → 0.3, WindowBefore → 0.8,
 WindowAfter → 1.38, SampleRate → 48 000, BandLow → 2000, BandHigh → 20 000,
 OutputSubfolder → mel_png_224_peak, ExportPNG → True, PlotPeaks → False,
 SkipIfExists → True, SaveLog → True, LogFile → encode_log.csv}

```

*此处的默认参数即为推荐使用的参数

这种设计的革命性在于，它将算法逻辑与实验参数彻底分离。我们无需修改一行核心代码，仅通过在顶层函数调用时传递不同的选项，就能快速地进行大量对比实验，例如测试不同窗长、不同频率范围对模型性能的影响。这种灵活性极大地降低了实验的门槛和时间成本，鼓励了探索和创新，使得整个特征提取流程可以被精确控制和迭代优化。

亮点六：端到端的智能自动化工作流 (EncodeAudioDatasetPeak)

函数 `EncodeAudioDatasetPeak` 是整个系统的大脑和指挥中心，它将所有独立的模块完美地串联起来，构建了一个从原始数据集到最终特征矩阵的、端到端的全自动化工作流。其设计的智能化和前瞻性，使其在处理大规模数据集时展现出革命性的效率和可靠性。其中，“`SkipIfExists`” -> `True` 选项的设计尤为关键。在处理成千上万个文件时，任务中断是常态。该选项使得任务可以随时被安全地停止和重启，系统在重启后会自动检测哪些文件的特征已经被生成，并智能地跳过这些已完成的工作，只处理剩余的部分。这为长时间、大规模的计算任务提供了断点续传的能力，节省了无法估量的机器时和人力。

此外，`Monitor[...]` 的使用为这个漫长的自动化过程提供了一个透明的进度条，让用户能够实时掌握任务进展。整个函数通过 `Check[...]` 语句包裹核心处理逻辑，确保了即使单个文件处理失败（例如文件损坏），也不会导致整个批量任务崩溃，而是会记录错误并继续处理下一个文件。这种集成了智能续跑、进度监控和错误容忍的自动化设计，是真正能从实验室原型走向工业化生产的标志。

亮点七：无缝的多领域统一计算环境

此项目展现了 Mathematica 作为一个统一计算平台的独特魅力和强大能力。在同一个环境中，代码无缝地融合了传统上需要不同专业库才能完成的领域：

1. **函数化编程的革命性调用**：只需通过XC-Index就可调取单一音频的特征频谱/进行多窗频谱生成，极大地方便了调试和可视化，极大提高了代码复用率


```
In[*]:= audioDataset["991443"]
```

```
Out[*]=
```



- 2. 数字信号处理:** 通过 `Audio[...]`, `BandpassFilter`, `AudioLocalMeasurements` 等内置函数, 实现了专业级的音频分析与操作, 提高了信噪比。
- 3. 机器学习与神经网络:** `NetEncoder[{"AudioMelSpectrogram", ...}]` 直接调用了底层的神经网络框架, 将复杂的时频变换封装成了一个简单的函数调用。

```
In[*]:= enc3 // FullForm
```

```
Out[*]//FullForm=
```

```
NetEncoder["AudioMelSpectrogram",
  Association[Rule["SampleRate", 48000], Rule["Normalization",
    NeuralNetworks`ValidatedParameter[None]], Rule["WindowSize", 1024],
  Rule["Offset", Quantity[8.333333333333332, "Milliseconds"]],
  Rule["WindowFunction", Automatic], Rule["MinimumFrequency", 2000],
  Rule["MaximumFrequency", 20000], Rule["NumberOfFilters", 224],
  Rule["Augmentation", NeuralNetworks`ValidatedParameter[None]],
  Rule["$PreEmphasis", None], Rule["TargetLength", 224],
  Rule["$TargetLength", 224], Rule["$Size", 224], Rule["$Version", "14.0.2"]],
  NeuralNetworks`TensorT[List[224, 224], NeuralNetworks`RealT]]
```

- 4. 数据可视化:** `autoFindAndPlotMelSpectrogramMulti` 能够准确清晰直观地展示出取样窗口, 便于调试分析与结果复现; `autoFindAndPlotMelSpectrogram` 和 `createAudioMatrixPlot` 能够即时地将抽象的矩阵和数据序列转化为直观的图形, 极大地便利了调试和分析过程。
- 5. 完整、鲁棒:** 如果键值对有误, 会捕获异常, 不会进行可能导致系统崩溃的运算, 并给出清晰可调试的错误信息。

```
In[*]:= autoFindAndPlotMelSpectrogram[audioDatase, 991443]
```

```
KeyExistsQ : 参数 audioDatase 不是一个有效的 Association 或规则列表.
```

```
[Warning] Audio "991443" not found in dataset or directory.
```

```
Out[*]=
```

```
$Failed
```

```
In[*]:= autoFindAndPlotMelSpectrogram[audioDataset, 9914435]
```

```
[Warning] Audio "9914435" not found in dataset or directory.
```

```
Out[*]=
```

```
$Failed
```

- 6. 系统 I/O 与文件管理:** `FileNameJoin`, `Export`, `FileExistsQ` 等函数提供了跨平台的、稳健的文件系统交互能力。

这种一站式的革命性体验, 消除了在不同编程语言或库之间导入导出、转换数据格式的巨大开销。所有数据, 无论是 `Audio` 对象、数值矩阵还是 `Graphics` 图形, 都在同一个符号体系下自由流转, 代码因此变得异常简洁、内聚且高效。这种内在的统一性, 让开发者可以将精力完全

集中在算法和逻辑本身。

亮点八. 全程可追溯的审计与日志系统

在自动化处理流程中，可追溯性是衡量其是否科学、可靠的关键标准。EncodeAudioDataset-Peak 函数内置了一个看似简单却至关重要的日志系统，为整个大规模实验提供了完整的审计轨迹。在处理完所有文件后，代码会通过 Export[logPath, Prepend[logRows, {"Key", ...}], "CSV"] 将一个详细的日志文件保存下来。这个日志文件精确地记录了每一个音频文件的处理结果：它的 Key（唯一标识符）、成功生成的窗口数量（“Windows”）、处理状态（“Status” - ‘ok’, ‘failed’, ‘skipped_exists’）、输出目录以及完成处理的时间戳。这个小小的 CSV 文件，在整个工作流中扮演了革命性的角色。它不仅是任务完成度的证明，更是问题排查的记录仪。当系统表现不佳时，研究人员可以轻易地通过日志追溯到哪些文件处理失败或产生了零窗口，从而进行针对性的修复。对于科学研究而言，这份日志是保证实验可复现性的关键文档。它将整个黑箱式的批量处理过程，变成了一个完全透明、有据可查的科学流程。

命令执行代码：

```
In[*]:= (*baseDir/enc3/audioDataset 已存在*)
baseDir = FileNameJoin[{$UserDocumentsDirectory, "XenoCantoDownloads"}];
resAssoc = EncodeAudioDatasetPeak[audioDataset,
  "NumWindows" → 5, "PeakMinDistance" → 0.3, "WindowBefore" → 0.8,
  "WindowAfter" → 1.38, "ExportPNG" → True, "PlotPeaks" → False,
  (*批量跑时建议关*) "SkipIfExists" → True, (*已有PNG则跳过*)
  "SaveLog" → True, "LogFile" → "encode_log_test_2.csv",
  "OutputSubfolder" → "mel_png_224_peak_test_with_subsubfolder"];

[Info] Created folder:
  /PathToUsersDocuments/XenoCantoDownloads/mel_png_224_peak_test_with_subsubfolder
[Info] Created folder:
  /PathToUsersDocuments/XenoCantoDownloads/mel_png_224_peak_test_with_subsubfolder/1
  Generated 5 peak windows for XC1000797

...

  Generated 5 peak windows for XC991443
  Generated 5 peak windows for XC996617
[Info] Saved log: /PathToUserDocuments/XenoCantoDownloads/encode_log_test_2.csv
[Done] Encoded keys: 416 / 416
(Wed 12 Nov 2025 18:00:09 → Wed 12 Nov 2025 18:09:44)
```

第五章：展望

本研究虽然已成功建立了一套从网络爬虫到声学特征生成的自动化流水线，但这仅是构建深港边境智能生态监测系统的第一步。基于当前生成的标准化、高质量梅尔频谱图（Mel Spectro-

gram) 数据集，未来的工作将重点围绕深度学习模型的构建、训练策略的优化以及生态学应用的落地展开。

5.1 基于 ResNet - 50 的构建声学图像分类模型并进行迁移学习

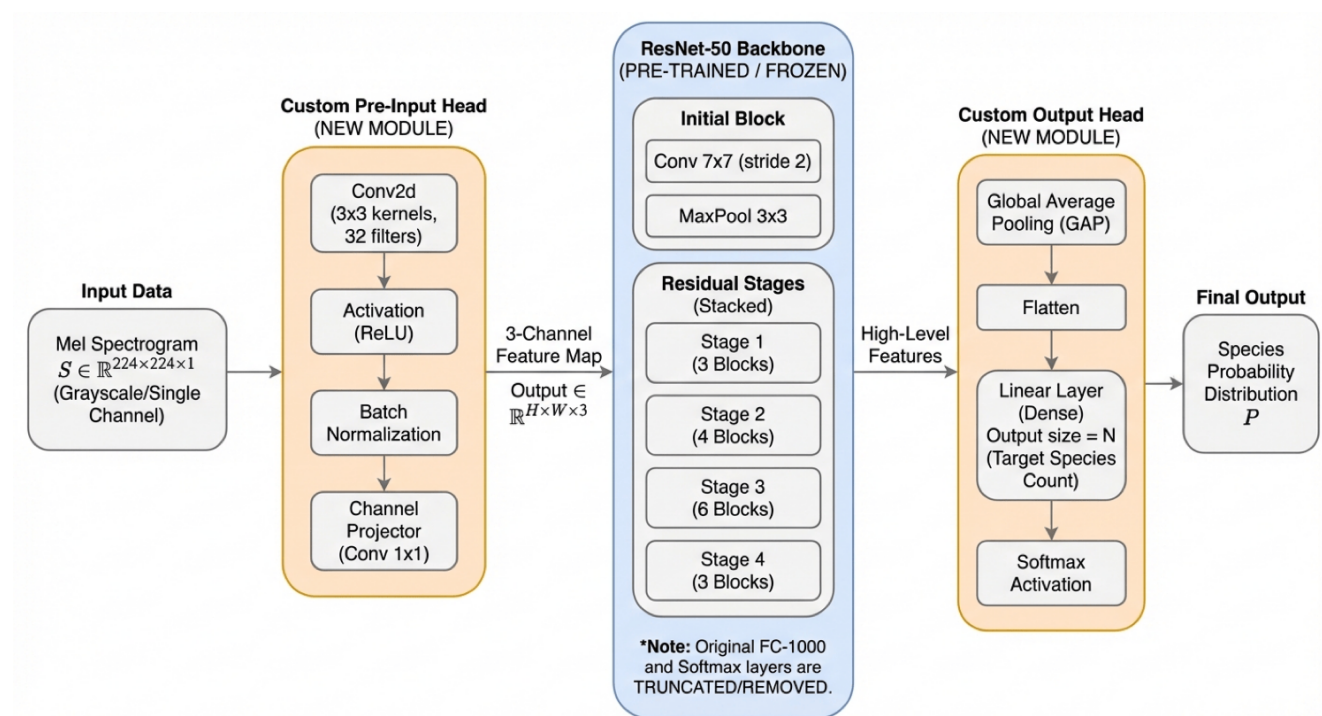
鉴于本研究已将非结构化的音频信号成功转化为 224*224 像素的标准化图像，下一阶段的核心任务是利用计算机视觉技术解决生物声学分类问题。我们将采用Mathematica预置 ResNet-50（残差网络）作为骨干网络，并针对声学频谱图的特性对模型架构进行以下关键性的定制化改造⁸。鉴于时间限制，该部分未包括在该项目中。
我们可以通过

NetModel["ResNet-50 Trained on ImageNet Competition Data"]

NetChain

	Input	image
conv1	ConvolutionLayer	array (size: 3 × 224 × 224)
bn_conv1	BatchNormalizationLayer	array (size: 64 × 112 × 112)
conv1_relu	Ramp	array (size: 64 × 112 × 112)
pool1_pad	PaddingLayer	array (size: 64 × 113 × 113)
pool1	PoolingLayer	array (size: 64 × 56 × 56)
2a	NetGraph (12 nodes)	array (size: 256 × 56 × 56)
2b	NetGraph (10 nodes)	array (size: 256 × 56 × 56)
2c	NetGraph (10 nodes)	array (size: 256 × 56 × 56)
3a	NetGraph (12 nodes)	array (size: 512 × 28 × 28)
3b	NetGraph (10 nodes)	array (size: 512 × 28 × 28)
3c	NetGraph (10 nodes)	array (size: 512 × 28 × 28)
3d	NetGraph (10 nodes)	array (size: 512 × 28 × 28)
4a	NetGraph (12 nodes)	array (size: 1024 × 14 × 14)
4b	NetGraph (10 nodes)	array (size: 1024 × 14 × 14)
4c	NetGraph (10 nodes)	array (size: 1024 × 14 × 14)
4d	NetGraph (10 nodes)	array (size: 1024 × 14 × 14)
4e	NetGraph (10 nodes)	array (size: 1024 × 14 × 14)
4f	NetGraph (10 nodes)	array (size: 1024 × 14 × 14)
5a	NetGraph (12 nodes)	array (size: 2048 × 7 × 7)
5b	NetGraph (10 nodes)	array (size: 2048 × 7 × 7)
5c	NetGraph (10 nodes)	array (size: 2048 × 7 × 7)
pool5	PoolingLayer	array (size: 2048 × 1 × 1)
flatten_0	FlattenLayer	vector (size: 2048)
fc1000	LinearLayer	vector (size: 1000)
prob	SoftmaxLayer	vector (size: 1000)
	Output	class

导入Mathematica预置的ResNet-50，并对其网络层进行如下调节，得到以下网络拓扑：

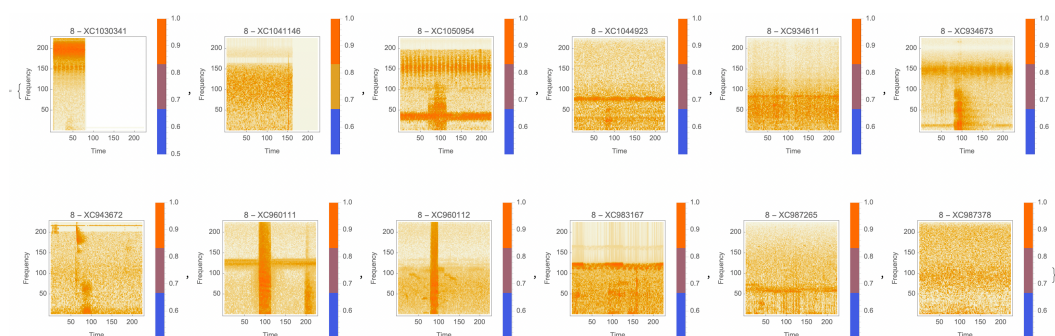


我们主要需要做的工作包括：

- 前端适配与特征增强：** 在 ResNet-50 的标准输入层之前，嵌入一个轻量级的卷积头。⁹ 考虑到原始 ResNet-50 通常针对 RGB 三通道自然图像设计，而梅尔频谱图本质上承载着时频强度的单通道或伪彩色信息，直接输入可能导致初始特征提取效率低下。该模块将包含小尺寸卷积核（3*3）与非线性激活函数，旨在起到两个作用：一是作为模态适配器，将单通道频谱数据映射到高维特征空间以匹配骨干网络的输入维度；二是作为去噪预处理器，在进入深层网络前滤除频谱图中可能存在的背景白噪声或瞬态干扰，强化鸟鸣信号的纹理边缘。
- 定制化分类输出头：** 移除 ResNet-50 原始针对 1000 类物体分类的全连接层及 Softmax 输出层，取而代之的是针对 N 个目标物种的线性分类层。该输出头将包含全局平均池化层以压缩空间信息，随后连接针对 N 个目标物种的线性分类层。通过这种架构调整，模型将能够把输入的声学指纹精准映射到具体的生物分类学标签。
- 骨干网络与迁移学习：** 利用在 ImageNet 上预训练的 ResNet-50 权重初始化模型主体参数，冻结中间卷积层的权重，仅微调高层语义特征提取层，可以有效利用深度网络强大的特征抽象能力，同时避免在特定鸟类样本较少的情况下发生过拟合。¹⁰

5.2 已知问题与鲁棒性增强：自适应信噪比筛选

正常鸣声指纹如：



A.1 按频谱形态分类的污染类型

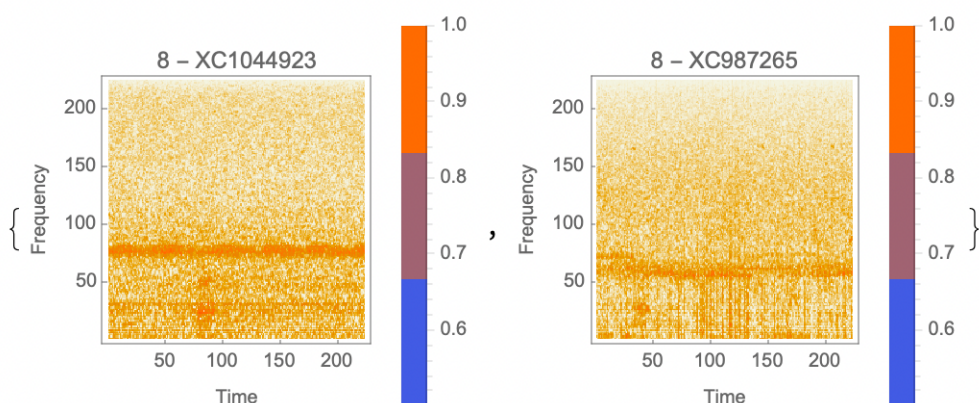
(A) 宽带持续噪声

表现：整条时间轴均有能量，频带大面积填满。

来源：风声、浪声、城市噪声、麦克风自噪。

特征：噪声和鸟声功率同级，主纹理难以分辨。

```
In[38]:= {autoFindAndPlotMelSpectrogram[audioDataset, 1044923],
           autoFindAndPlotMelSpectrogram[audioDataset, 987265]}
```

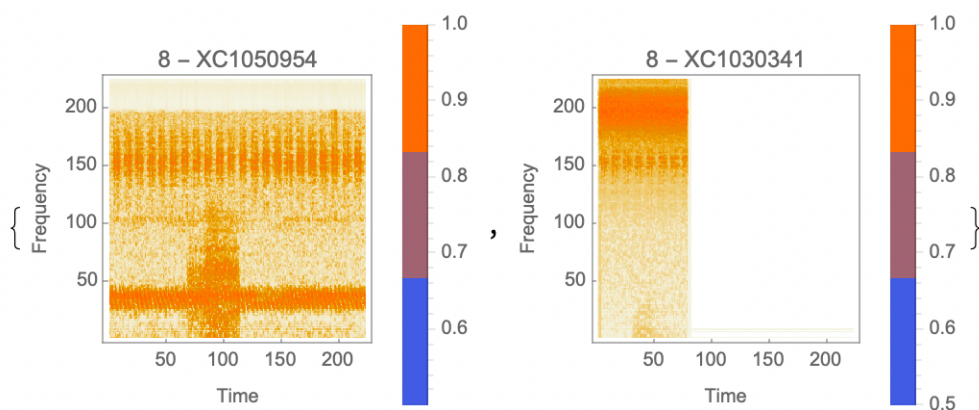


(B) 强窄带+多谐波干扰

表现：多条规则水平纹理贯穿频带，明显是其它鸣禽或昆虫的歌声。

结果：自动截片把“别的鸟”当成主角，Purple_Heron 反而变成背景。

```
In[39]:= {autoFindAndPlotMelSpectrogram[audioDataset, 1050954],
           autoFindAndPlotMelSpectrogram[audioDataset, 1030341]}
```

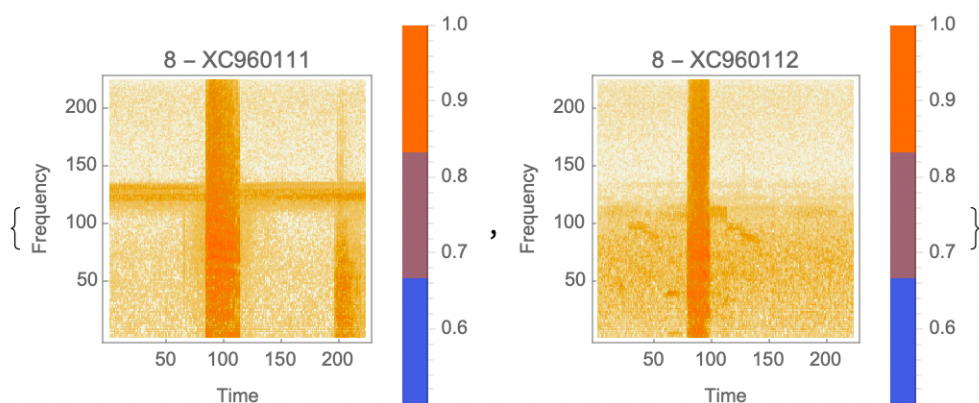
(C) 瞬时冲击噪声

表现：竖直能量柱，整频带瞬时抬升。

来源：人声、踩踏、设备碰撞、水声。

特点：一旦被选为窗口中心，整个 Mel 图的动态范围被撑满。

```
In[40]:= {autoFindAndPlotMelSpectrogram[audioDataset, 960 111],
          autoFindAndPlotMelSpectrogram[audioDataset, 960 112]}
```

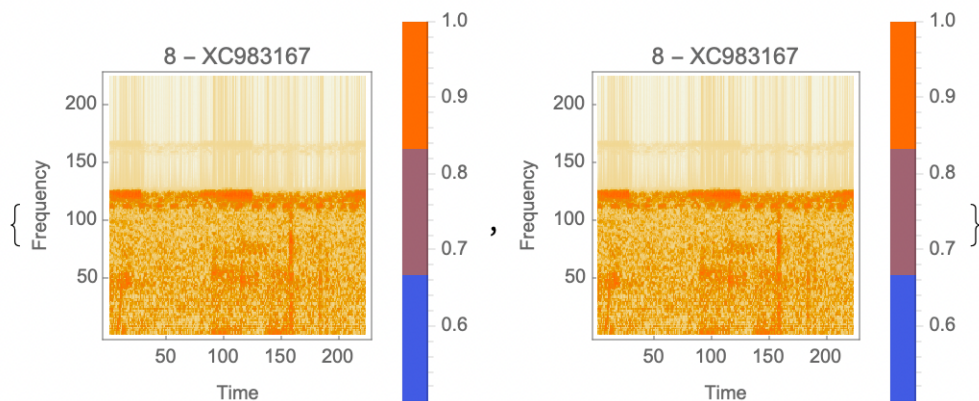


(D) 类白噪声样本

表现：能量均匀分布、无明显结构。

结果：视觉上几乎无法识别物种，等价于“标签噪声”。

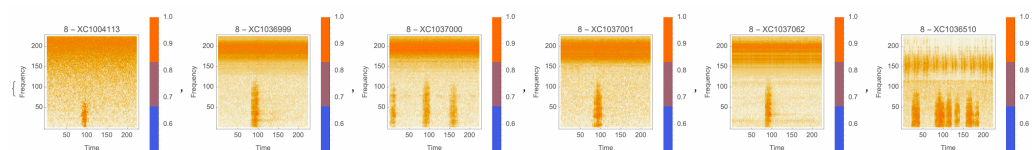
```
In[41]:= {autoFindAndPlotMelSpectrogram[audioDataset, 983 167],
          autoFindAndPlotMelSpectrogram[audioDataset, 983 167]}
```



面对网络采集音频中不可避免的环境噪声干扰，单纯依赖模型自身的抗噪能力是不够的。未来的工作将构建一套基于图像统计学的自适应信噪比（SNR）筛选机制，作为ResNet - 50 输入前的质量守门员。

B. 来自8 Purple_Heron (草鹭) 的高频噪音污染数据集

```
In[ ]:= setHighFPolluted = {1004113, 1036999, 1037000, 1037001, 1037062, 1036510};
Table[autoFindAndPlotMelSpectrogram[audioDataset, x], {x, setHighFPolluted}]
```



该物种对应的子数据集中存在大量高频能量异常富集、整幅频谱被背景噪声覆盖的样本。其典型表现为：

1. 高频 Mel 频带出现大片连续能量分布；
2. 存在清晰、稳定的鸟鸣时频纹理；
3. 多数样本在时间轴上无明显 鸣唱事件 结构；
4. 频谱对比度低，主信号与背景噪声难以区分。

这表明该数据子集存在系统性的高频噪音污染问题，并不属于个别异常样本，而是具有数据源级别的偏差特征。我们可能可以利用ResNet的卷积特性进行应对，减小其在后续自动化识别流程中的影响。

C. 问题的来源



草鹭¹¹

1. 湿地环境是个多源噪声叠加场，包含昆虫、蛙类、风、水、远处交通等声源；
2. 草鹭发声低频、短促、稀疏，经常远距离录音 导致录音信噪比（SNR）低；
3. Xeno-Canto 出于动物保护原因，不允许我们下载该物种的大部分A评级录音，导致选取了大量质量较低录音（但其实该物种目前的保护状况是“无危”级别，并未受威胁...）
4. 自动机制通过响度指标优先抓取，在极低SNR数据集中表现较差；

D.解决的可能路径

1. **低质量样本的自动熔断：** 我们将利用 Wolfram 语言的图像处理算法，计算 224*224 梅尔频谱图的熵值与像素强度分布。对于那些背景噪声掩盖了鸟鸣纹理、导致信噪比过低的“失败”频谱图，系统将自动执行跳过或标记处理。这不仅能防止无效数据污染梯度下降过程，还能为模型留出“拒识”空间，即当输入信号质量不足以做出判断时，模型应倾向于不输出结果而非强行分类。
2. **对抗性数据增强：** 在清洗后的高置信度数据集上，我们将引入时间/频率掩蔽技术。通过随机遮蔽频谱图的特定频段或时间片，模拟野外鸟鸣被风声、虫鸣部分遮盖的场景，迫使模型学习鸟鸣声纹的全局拓扑结构而非局部像素特征，从而显著提升模型在深港边境复杂声环境下的鲁棒性。

5.3. 种内多模态特征的细粒度挖掘

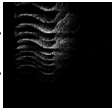
鸟类发声具有高度的复杂性，单一物种往往具备多种不同的鸣唱模式或叫声。简单的物种层级分类可能掩盖了具有重要生态学意义的行为信息。

一下显示的是输出PNG，纵轴（频率对数）方向朝下，横轴（时间）朝右：

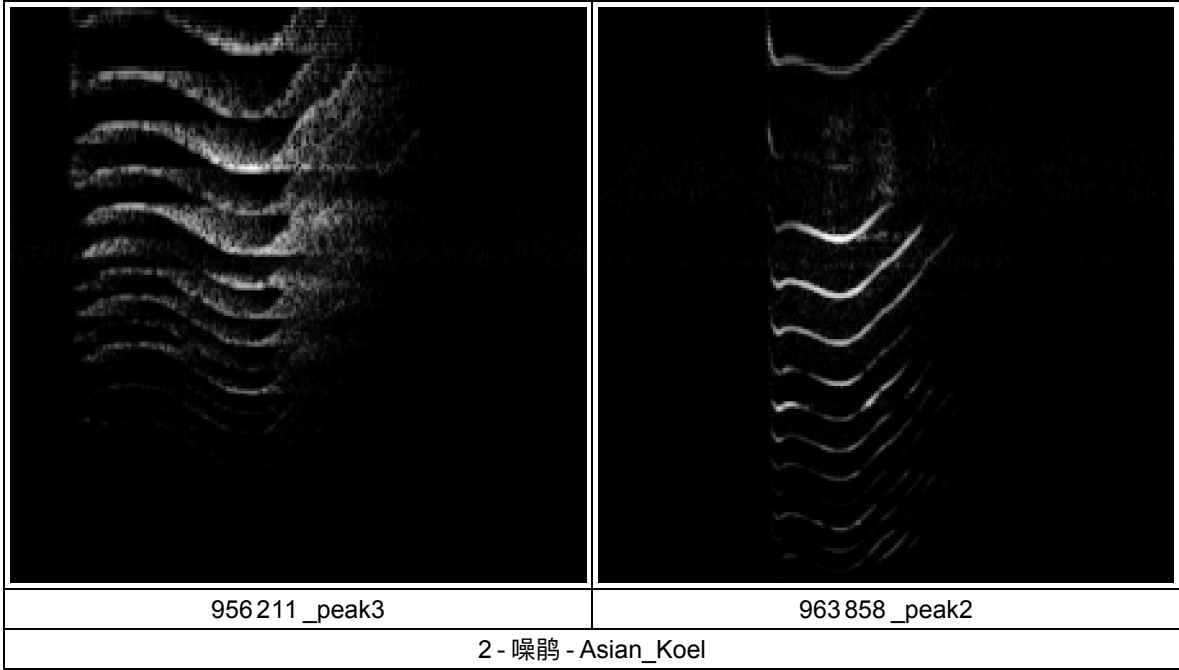
同种鸟的语音存在音高变换：



噪鵲(雄性) Koel (Male)¹²

```
In[*]:= Grid[{{Image[, ImageSize -> 300], Image[, ImageSize -> 300]},  
  {Text[956 211 _peak3], Text[963 858 _peak2]},  
  {Text[2 - Asian_Koel - (噪鹛)], SpanFromLeft}}, Frame -> All]
```

Out[*]=



同种鸟存在不同鸣唱方式：



效的编程、音频信号处理与可视化能力，更重要的是，它所倡导的函数式与符号化思维方式，深刻改变了我对计算的思考方式；
最后，谨以此文，献给所有给予我支持、启发与陪伴的人。

References

- 1 <https://eaaflyway.net/the-flyway/>
- 2 Wikramanayake, E., Or, C., Costa, F., Wen, X., Cheung, F., & Shapiro, A. (2020). A climate adaptation strategy for Mai Po Inner Deep Bay Ramsar site : Steppingstone to climate proofing the East Asian – Australasian Flyway. PLOS ONE, 15 (10), e0239945. <https://doi.org/10.1371/journal.pone.0239945>
- 3 <https://cco.gov.hk/tc/project-sites/mai-po-nature-reserve>
- 4 Morales, G., Vargas, V., Espejo, D., Poblete, V., Tomasevic, J. A., Otondo, F., & Navedo, J. G. (2022). Method for passive acoustic monitoring of bird communities using UMAP and a deep neural network. Ecological Informatics, 72, 101909. <https://doi.org/10.1016/j.ecoinf.2022.101909>
- 5 Liu, Y., Zollinger, S. A., & Brumm, H. (2020). Chronic exposure to urban noise during the vocal learning period does not lead to increased song frequencies in zebra finches. Behavioral Ecology and Sociobiology, 75 (1), 3. <https://doi.org/10.1007/s00265-020-02942-w>
- 6 Slabbekoorn, H., & Peet, M. (2003). Birds sing at a higher pitch in urban noise. Nature, 424 (6946), 267–267. <https://doi.org/10.1038/424267a>
- 7 Wang, H., Xu, Y., Yu, Y., Lin, Y., & Ran, J. (2022). An Efficient Model for a Vast Number of Bird Species Identification Based on Acoustic Features. Animals, 12 (18), 2434. <https://doi.org/10.3390/ani12182434>
- 8 Sankupellay, M., & Konovalov, D. (2018). Bird Call Recognition using Deep Convolutional Neural Network, ResNet – 50.
- 9 Zeghidour, N., Teboul, O., Quitry, F. de C., & Tagliasacchi, M. (2021). LEAF : A Learnable Frontend for Audio Classification (No. arXiv : 2101.08596). arXiv. <https://doi.org/10.48550/arXiv.2101.08596>
- 10 Efremova, D. B., Sankupellay, M., & Konovalov, D. A. (2019). Data – Efficient Classification of Birdcall Through Convolutional Neural Networks Transfer Learning (No. arXiv : 1909.07526). arXiv. <https://doi.org/10.48550/arXiv.1909.07526>
- 11 <https://www.iam.gov.mo/nature/c/biologyInfo/bird/detail?id=2cd24b93-cbf0-431c-aca4-9d850c330df8>
- 12 https://www.lcsd.gov.hk/tc/green/birds/overview/bird_5.html
- 13 <https://www.wetlandpark.gov.hk/tc/biodiversity/beauty-of-wetlands/wildlife/prinia-inornata>