arXiv:2303.01665v1 [cs.SD] 3 Mar 2023

LooperGP：一种利用 GuitarPro 六线谱进行现场编程演出的可循环序列模型

Sara Adkins[1],Pedro Sarmento[1][0000−0002−4518−0194]，以及 Mathieu Barthet[1][0000−0002−9869−1668]

英国伦敦玛丽女王大学数字音乐中心 sara.adkins65@gmail.com，{p.p.sarmento, m.barthet}@qmul.ac.uk

摘要。 尽管深度学习模型在离线任务中取得了令人瞩目的成果，但由于缺少具有音乐意义的控制参数，且生成结果缺乏结构化的音乐形式，这类符号音乐生成模型尚未广泛应用于现场表演。为解决这些问题，我们提出 LooperGP，这是一种引导 Transformer-XL 模型生成指定小节数和拍号的可循环乐句的方法，可用于现场编程演出的工具。我们证明，通过在从 DadaGP 数据集中提取的 93,681 个音乐循环片段上训练 LooperGP [22], ，我们能够引导其生成模型输出的可循环乐句数量达到基线的 3 倍。在一项由 31 名参与者完成的主观聆听测试中，LooperGP 循环片段在原创性、音乐连贯性和循环流畅度方面的评分中位数均为正，展现了其作为演出工具的潜力。

关键词： 可控音乐生成 · 序列模型 · 现场编程 · Transformer · 人工智能音乐 · 循环片段 · 吉他谱

1 引言

诸如流行音乐Transformer [11] 和 Transformer-XL[8] 这样的序列模型，能够生成符号化复调音乐，并在数分钟的音频中保持连贯的音乐结构。然而，这类模型存在若干问题，使其难以适用于现场表演。它们无法控制调号、拍号、乐器编制、音乐流派和时长等重要音乐参数。尽管生成的内容在音乐上可能听起来合理，但如果这些音乐结构参数无法控制，就很难将其融入演出中。此外，推理速度缓慢，使得在传统演出场景中开发可用的序列模型操作界面变得十分困难。

现场编程是一种艺术形式，表演者编写代码并实时合成音乐；它有潜力成为序列模型可行的表演环境。现场编程语言可以被视为一种乐谱记谱工具，编辑内容会被实时评估 [14]。在典型的表演中，表演者会通过算法作曲技术生成符号模式 [6] 或硬编码来修改乐谱。该模式

随后通常会被加入音频混音中，并作为循环重复播放 [16]。不同于传统的模式生成技术，可以使用序列模型生成模式选项，供表演者选择。

本文通过提出一种训练与推理算法，解决可控生成音乐的问题，引导Transformer模型生成具有指定长度、调和拍号的可循环乐句。随后，我们将生成的可行循环数量与基线模型进行比较，以评估我们的循环模型（称为LooperGP）的效果。最后，我们呈现一项听力测试的结果，通过比较参与者对人类创作循环与机器生成循环在创造性、喜爱度、连贯性和平滑度方面的评分，对 LooperGP 进行主观评估，并确定有待改进的方面。我们的循环模型和推理控制旨在通过解决确保模型输出符合持续进行的音乐表演情境这一限制，促进深度生成模型在现场编程环境中的应用。未来工作将探索 LooperGP 在现场表演中的评估。

2 相关工作

2.1 用于音乐生成的序列模型

包括循环神经网络、变分自编码器和生成对抗网络在内的各种深度学习架构，都已被用于生成符号音乐 [5], [12], ，而 Transformer 在建模时间依赖关系方面尤其成功 [8]。正如 Mu- sic Transformer 在 [11], 中所展示的，Transformer 架构的自注意力机制使模型能够回溯此前生成的动机和乐句，使其成为符号音乐生成中颇具潜力的架构。流行音乐Transformer 论文进一步表明，音乐数据的存储方式会显著影响输出质量。例如， [11] 的结果表明，将节拍位置等音乐知识编码进训练数据，能够带来更好的节拍和拍号显著性。

2.2 可控音乐生成

阻碍符号音乐生成系统在主流音乐创作中普及的一大障碍，是缺乏控制能力。通常，唯一可用的控制方式是温度参数 [2],，用于从模型的概率分布中选择一个词元。DeepBach 系统 [9], 是一个例外，它允许用户在巴赫风格圣咏曲中固定特定音符，从而对生成过程施加约束。在这些音符约束下，双向长短期记忆网络（LSTM）模型通过吉布斯采样补全圣咏曲的其余部分。在现场编程场景中，这种程度的定制有助于生成具有良好循环收尾效果的乐句，因为可以将起始音符和结束音符硬编码。不过，DeepBach 方法受限于固定的感受野，而且只在巴赫众赞歌上进行过测试，因此可能难以用于现代音乐流派的现场表演。

原始Transformer通常会围绕作为提示输入的动机进行扩展，但这种行为并不能得到保证。 [23] 中的主题Transformer提出了一种新的模型架构，可根据特定动机/主题对Transformer编码器进行条件化。训练数据经过处理后，会通过将相似小节聚类在一起，从每个训练样本中提取“主题-乐曲对”。通过在这些配对上训练，模型能够学习如何在乐曲中扩展动机，并因重复和变化主题而获得奖励。在现场表演中，这种扩展现有素材的能力十分实用，因为它能确保生成的内容与已有音乐素材相关。

2.3 现场编程

[20] 将现场编程定义为“在音乐会环境下对计算机进行编程的行为”，而我们特别关注能够生成音乐输出的程序。传统音乐表演以逐音符或逐和弦的方式指定音乐流，而现场编程则通过对音乐创作进行乐谱级控制，在更高的抽象层次上进行表演 [20]。现场编程实践的一大重点是控制生成过程：表演者控制生成算法的参数，以产生音乐输出，使编写软件成为一种表演行为[6]。

将深度学习（DL）融入现场编程演出的先前研究包括 Cibo [24], ，这是一种基于文本的序列到序列模型，可生成 Tidal Cycles [16] 代码；以及 RaveForce [13], ，这是一条使用开放声音控制（OSC）将从深度学习模型中推导出的合成器参数传输至 SuperCollider 服务器的处理流程 [15]。

2.4 DadaGP 数据集

在现场编程场景中使用的生成模型，理想情况下应能够生成多种现代音乐流派的音乐，以契合这类演出的典型风格。DadaGP 数据集 [22]包含超过 26,000 首歌曲，涵盖 700 种音乐流派；其专用编码器则将歌曲表示为兼容序列模型的词元。该数据集以摇滚乐和金属乐为主，同时也包含爵士乐、Pop、古典音乐和电子舞曲（EDM）。此外，词元集合还纳入了弦乐器的表现性演奏信息，例如颤音、掌闷音和击弦/勾弦，因此相较于 MIDI 更具优势。DadaGP 数据集丰富的歌曲多样性和富有表现力的词元，使其成为现场编程演出音乐生成的理想训练语料库候选。

[22] 还介绍了一个在 DadaGP 数据集上训练的 Transformer-XL 模型，该模型有 8 个注意力头和 12 个自注意力头。我们将此模型用作基线，并以此为起点构建下一节介绍的 LooperGP 模型。

3 方法论

[23]中的主题Transformer展现出良好效果，证明了通过重新组织训练数据，可以引导模型生成特定类型的输出。流行音乐

[11] 中的 Music Transformer 表明，在词元中加入拍位置、小节位置和和弦标签等音乐信息检索（MIR）特征，可以改善生成内容的高层次音乐特征，例如拍显著性。结合这些思路，我们从训练数据中提取“可循环乐句”，构建一个仅由能够自然循环的片段组成的新训练语料库。每个循环都将存储为有序的DadaGP标记列表 [22], ，其中的标记描述了循环中每个激活乐器的乐谱和演奏细节。我们的目标是在推理过程中进一步引导模型生成可循环乐句，以便将其融入现场编程演出。

3.1 定义循环片段

以往关于循环提取的研究将循环片段定义为一段在重复播放时能够无缝衔接的短音频 [21], [7]。无缝的“可循环性”最终取决于听众的主观感受，但我们可以关注作品的结构属性，以识别可重复的部分。就我们的目的而言，如果歌曲的某个片段以一段达到最短长度要求的重复乐句作为首尾边界，我们就认为该片段可循环播放[1]。图 1 展示了 AC/DC 的《You Shook Me All Night Long》中的一个循环片段。

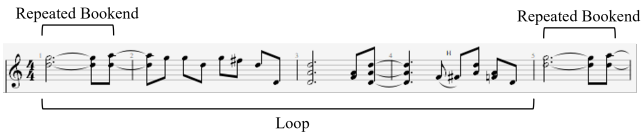

图 1. AC/DC 的《You Shook Me All Night Long》中的一个循环片段示例。图中仅展示吉他声部，不过该循环片段在 DadaGP 数据集中是多轨的。

3.2 提取循环片段

为了识别给定歌曲中的所有循环片段，我们必须先识别所有重复乐句，或如图 1 所示的“端部重复乐句”。为了从给定歌曲中提取重复段，我们采用 [10] 中提出的相关矩阵方法。DadaGP 数据集 [22] 以文本格式提供，该格式专为多轨六线谱编辑软件 Guitar Pro 设计。首先将 Guitar Pro 格式的歌曲转换为一个激活音符集合列表 M，如公式 1 所示。 $N(t,\ d)$ 表示在刻度时间 t 时所有激活音符的集合，其时长为 d，使得 $t_0 = 0$ 且 $t_n = t_{n-1} + d_{n-1}$。

$$M = [N(t_0, d_0), N(t_1, d_0), ..., N(t_n, d_n)] \tag{1}$$

[1]这里我们关注内容完全相同的重复循环片段，但值得注意的是，更广义的定义也可以涵盖在重复段之间出现某些类型音乐变体的循环片段（例如转调）。

设 C_s 和 D_s 为歌曲 s 在旋律线 M_s 上的 $L \times L$ 矩阵，其中 L 是 M_s 的长度。C_s 被构建为音符的相关矩阵，如公式 2 所示，其中 $C_s(i,\ j)$ 表示在 $M_s[i]$ 和 $M_s[j]$ 处结束的重复片段中音符的数量。类似地，$D_s(i,\ j)$ 是一个相关矩阵，用于存储在 $M_s[i]$ 和 $M_s[j]$ 处结束的重复片段的时长（以时钟刻度计）。

$$C_s[i,j] = C_s[i-1,j-1] + 1 \text{ if } M_s[i] = M_s[j] \text{ else } 0 \tag{2}$$

矩阵 C_s 和 D_s 可帮助我们识别循环候选项，并根据时长和音符长度筛选重复乐句。相关矩阵的递归特性确保，对于任意给定的单元格 $C_s[i,j] > 0$; 均有$C_s[i-1,j-1] = C_s[i,j] - 1$。因此，我们可以在相关矩阵中查找满足 $C_s[i,j] = L_{min}$的单元格，以确定所有长度至少为 L_{min} 的重复段的起始点。重复乐句$[s_i : s_j]$ 的两个起始索引根据公式 3 计算，并构成一个潜在循环 $p_{i,\ j}$，如公式 4 所定义。

$$[s_i, s_j] = (i - L_{min}, j - L_{min}) \text{ iff } C_s[i,j] = L_{min} \tag{3}$$

$$p_{i,j} = M[s_i : s_j] \tag{4}$$

给定所有潜在循环 $p_{i,\ j}$ 及其对应端点$[s_i, s_j]$的列表，我们可以筛除无意义、过长或端部重复乐句过短的循环。以下长度参数可配置：

– 最小重复音符数（L_{min}）：端部乐句中的最少音符数– 最小重复拍数（RD_{min}）：端部乐句中的最少拍数– 最小循环小节数（LB_{min}）：两个端部乐句之间的最少小节数– 最大循环小节数（LB_{max}）：两个端部乐句之间的最大小节数

我们通过比较 $M[s_i]$ 和 $M[s_j]$之间的时间戳差，并利用歌曲的拍号元数据将此时钟刻度时长转换为小节数，以筛选 LB_{min} 和 LB_{max}。接下来，我们通过定位每个循环首尾段的末尾来筛选 RD_{min}。对于端点为 $[s_i, s_j]$的循环，我们可以沿着C_s 的对角线遍历，直到遇到单元格值减小，如方程 5 所示，从而找到重复子乐句的末尾。这个减小表明 s_i 和 s_j 之间的匹配已经结束，我们也就找到了首尾段的末尾。然后，我们可以使用方程 6 查找首尾段的时钟刻度时长，并筛除时长短于所需值的循环。

$$n_{end} = n | (C_s[i+n, j+n] < C_s[i+n-1, j+n-1]) \tag{5}$$

$$D_{ticks} = D_s[i + n_{end} - 1, j + n_{end} - 1] \tag{6}$$

提升提取出的循环片段音乐趣味性的最后一种筛选策略，是按音符密度筛选，以过滤掉主要由休止符或长时延音构成的循环片段。我们将循环片段 $p_{i,\ j}$ 的音符密度（ρ）定义为每小节音符起始的平均数量，并按乐器音轨的数量进行缩放。这

一定义在公式 7 中形式化表示，其中 B 是该循环片段中的小节数， T 是乐器轨道的数量。例如，$p_{i,\ j}$ 中 ρ 的值为 4.0，意味着每条音轨平均每小节有四个音符起始。

$$\rho_{p_{i,j}} = len(p_{i,j}) \times \frac{T}{B} \tag{7}$$

除了使用相关矩阵算法提取循环片段外，我们还可以通过在训练语料库中搜索反复记号并按拍数时长筛选，来识别内置循环片段。完整的循环数据集由提取出的循环片段和内置循环片段组成。表 1 展示了循环提取参数的不同组合如何影响识别出的循环片段数量。我们选择了加粗显示的参数设置（共 93,681 个循环片段），因为在所有尝试过的参数组合中，它对应的循环片段数量处于中位数。

表 1. 循环提取参数统计。

小节	密度 L	最小值 RD	最小值	循环片段	每首歌曲平均循环片段数
4-8	4	4	4	79,066	3.02
4-8	3	4	4	80,557	3.008
4-8	4	2	4	82,244	3.14
4-8	3	2	4	83,787	3.20
4-16	4	4	4	88,842	3.34
4-16	3	4	4	90,512	3.46
4-8	4	4	2	92,325	3.52
4-16	4	2	4	92,422	3.53
4-8	**3**	**4**	**2**	**93,681**	**3.59**
4-16	3	2	4	94,138	3.60
4-8	4	2	2	99,331	3.79
4-8	3	2	2	100,930	3.89
4-16	4	4	2	104,910	4.01
4-16	3	4	2	106,607	4.08
4-16	4	2	2	113,547	4.34
4-16	3	2	2	115,347	4.41

表2.循环数据集统计。

数据格式	文件	每首歌曲最大词元数	每首歌曲平均词元数	Size
Hard	66,637	12,253	1,376	9.13GB
禁止	23,024	43,954	2,270	5.19GB
DadaGP	26,158	51,997	4,456	9.56GB

3.3 训练

我们尝试了两种不同的循环训练数据集格式。第一种格式（“小节重复”）在给定歌曲的每个循环片段两侧分别加上 repeat_open 和 repeat_close 词元，然后将这些禁止循环片段连接到同一个文件中。第二种格式（“硬重复”）将每个循环片段存储在单独的文件中，并手动将该循环片段重复 4 次。我们假设，通过采用小节重复数据集格式进行训练，模型将学会在生成过程中确定 repeat open/close 词元的最佳放置位置，从而形成音乐上连贯的循环片段。

硬重复模型可能会由于每个循环片段的多次重复而更好地学习平滑过渡。然而，如表2所示，硬重复存储格式的空间效率远低于小节重复，因此在训练过程中完成一个训练轮次所需的时间也更长。我们采用 [22] 中介绍的流行音乐 Transformer-XL架构来训练 LooperGP。对于每种数据格式，我们都从头训练了一个模型，并以 DadaGP 模型的第 200 个训练轮次为起点训练了第二个模型。预训练的 DadaGP 模型已经学会了如何创作结构连贯的音乐，但早期测试显示，它往往会随机切换到不同的主题段落。通过进一步在循环数据集上训练该模型，我们希望使模型学会将生成结果偏向于连贯的循环片段，然后利用第 3.2 节介绍的过滤流程将这些片段提取出来。不同训练配置的结果将在结果部分中介绍。对于全部四个模型，我们使用了 0.0001 的学习率、15% 的随机失活，以及 [22] 所采用的 Adam 优化器。

3.4 可控推理

在现场编程演出中，表演者能够控制音乐生成的音乐参数非常重要。为了让生成循环融入正在进行的演出，循环必须与当前演奏内容的调和拍号一致。控制循环的长度（以小节计）也至关重要，以确保乐句与演出组曲中当前正在演奏的其他模式对齐（或有意不对齐）。

[22] 观察到，使用一个或多个来自目标乐器、构成该调根和弦的音符来预置推理，通常能够成功生成该调和相应乐器编制的音乐，至少在较短时长内如此。图 2 展示了以鼓组（无音高）、吉他（A3）和贝斯（A2）的一个八分音符作为预置输入后，模型如何沿用既定的乐器编制，以 A小调继续生成数个小节。

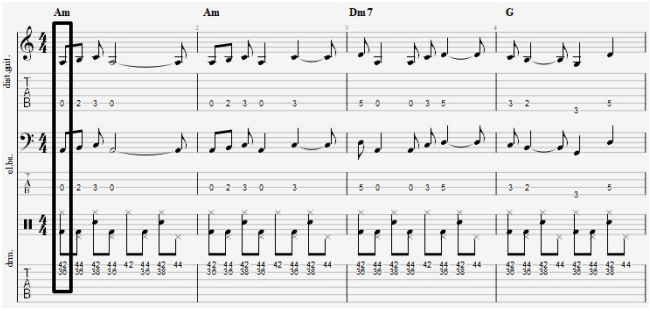

图 2。使用 DadaGP 模型生成的示例输出，黑框中显示了预置输入。和弦符号由人工分析得出。

我们进一步探索了如何在推理过程中控制拍号和时长。DadaGPTransformermodel 的运行方式是生成指定数量的词元 [22], ，但这些词元与时长没有任何关系。为了解决

这种不可预测性，我们修改了推理方法，通过累加 wait:xxx 标记的时钟刻度值来跟踪累计时长，并在达到指定时钟刻度数后人为插入新的_小节词元，以确保小节时长。如果 wait 标记的时长跨越小节边界，我们就缩短其时长，使其恰好落在小节内。我们还累计统计新_小节词元的数量，并在达到所需小节数时人为插入一个“end”词元，以替换最后一个新_小节词元。这种方法的一个缺点是，无法区分小节时长相同的不同拍号，例如 3/4 和 6/8。

LooperGP 生成所需数量的小节后，下一步是使用我们的过滤流程提取循环片段。通过将 LB_{min} 和 LB_{max}参数设为相同的值，我们可以指定要输出的循环片段的确切长度。由于 LooperGP 专门使用循环片段数据进行训练，因此与基线模型相比，生成的输出应包含显著更多的循环片段。我们在结果部分比较 LooperGP 不同版本与基线 DadaGP 模型的循环片段生成率。

4 结果

4.1 训练表现

第 3.3 节中所述的四种模型配置分别训练了 20 个训练轮次（受资源限制），并在循环数据集上采用 85/15 的训练/验证集划分。图 3 展示了交叉熵训练损失和验证损失曲线。正如预期，预训练模型配置的损失曲线比从头训练的模型平缓得多，因为它们已经在原始 DadaGP 数据集上收敛。除预训练 Barred Repeats 模型外，所有模型的验证损失都会随时间下降。

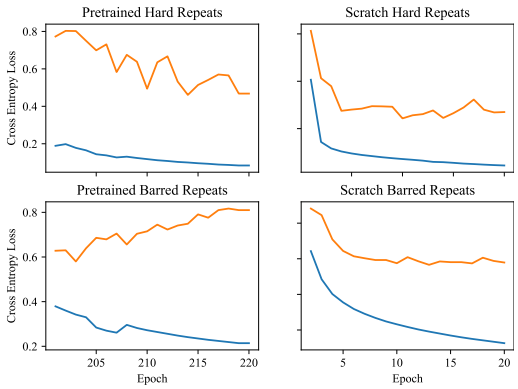

图 3. 4 种 LooperGP 模型配置在 20 个训练轮次中的训练损失（蓝色）和验证损失（橙色）

4.2 循环性能

为了评估各模型配置生成输出的质量，我们从每个模型生成了 25 个片段，并计算每个片段提取出的循环片段平均数量，以及根据公式 7 定义的平均循环音符密度。每个模型均配置为生成 16 个 4/4 拍音乐小节，并使用从训练数据中提取的随机速度和乐器配置作为初始条件。随后，我们使用第 3.2 节中的循环提取算法，并采用以下参数： $L_{min} = 4$、 $RD_{min} = 2$、 $LB_{min} = 4$ 和 $LB_{max} = 4$，这些参数取自表 1 的中位数

推理分析的结果如表 3 所示。显而易见，从头训练的 Hard Repeats 模型无法生成有意义的循环片段；提取出的循环片段非常少，平均音符密度也极低。通过人工查看该模型生成的结果发现，大多数输出由长延音和过多的休止符构成，这表明它未能充分学习音乐乐句。预训练的 Hard Loops 模型不存在休止符过多的问题，但其表现仍未优于基线 DadaGP 模型，在总共 25 个片段中仅多生成了 1 个有效循环。

表 3. 模型间的循环提取统计。

模型	找到的循环片段数	循环片段均值	音符密度均值
DadaGP 基线	20	0.80	6.03
预训练硬盘	21	0.84	6.84
从头训练硬盘	3	0.12	0.25
预训练禁止	88	3.52	6.96
从头训练禁止	77	3.08	9.80

Barred Repeats 模型生成的循环片段数量均远高于基线。在这两种情况下，每生成 16 小节的内容，都能提取出三个或更多四小节循环，且平均音符密度较高。尽管这些结果令人鼓舞，但它们并不能说明提取出的循环片段在音乐质量方面如何。为准备主观聆听测试的刺激材料，我们继续训练预训练 Barred Repeats 模型，并在第 40 轮训练后停止，此时交叉熵损失为 0.148。

4.3 主观评估

为了评估 LooperGP 生成循环的主观质量，我们开展了一项在线听力研究，请参与者评估机器生成的循环乐段，以及从训练数据中提取的人工创作的循环乐段。参与者聆听了 50 段片段，每段都包含一个重复四次的四小节循环乐段。其中 25 段为人工创作，另外 25 段为机器生成[2]。这些片段以随机顺序播放给参与者，且不透露生成方法，

[2]听力测试片段链接：https://drive.google.com/drive/folders/1I0MCPYjj8nXqKkmDN-d-C2ETOHJpCZyn?usp=share_link

参与者还需回答以下关于每段片段的问题，并填写 Goldsmiths 音乐熟练度指数（MSI）问卷 [19]:

1. 你以前听过这段音乐吗？（是/否） 2. 你认为这个循环片段中的音乐是由人类还是机器创作的？（人类/机器） 3. 请评价你对以下陈述的同意或不同意程度：（7 点李克特量表）Q1：这个循环片段中的音乐具有原创性和音乐创意。Q2：我喜欢这个循环片段中的音乐。Q3：在这个循环片段中，音乐的节奏、旋律和和声连贯。Q4：重复段之间的过渡在节奏、旋律和和声方面都很流畅。4. 请简要说明你评分的理由。你可以列出关键词，不必写完整句子。

听力测试片段 25 段人类创作的循环片段是从 DadaGP 训练数据中提取的所有四小节循环乐段中随机抽取的。其余 25 段片段由 LooperGP 模型生成。生成片段时，先从 DadaGP 数据集中四小节循环乐段中随机抽取一个初始速度、乐器编制和第一个音符，然后进行 16 小节 4/4拍的推理。接着，从生成结果中提取所有四小节循环乐段，并随机选取其中一段作为最终片段。此过程重复 25 次。所有片段均使用 GuitarPro 7 中的默认虚拟乐器渲染为 MP3（可变码率 260 kbps）。参与者被要求在回答问题时，专注于音乐创作的质量，而非虚拟乐器的音质或音乐制作混音的质量。

参与者 我们共招募了 31 名参与者，其中 6 人通过面向整个系的参与者招募邮件报名，另外 25 人来自数据收集网站 Prolific。纳入标准为：能够使用耳机、听力正常、没有失乐症（音盲），并且以音乐为爱好。参与者的性别分布大致均衡，其中男性 17 人，女性 14 人。

序数数据转换 问题3中的四个陈述收集到的李克特量表数据属于序数数据。关于李克特数据能否在统计检验中视为连续数据，学界尚有不同看法（[4] 和[25]）。均值不一定是有用的指标，因为参与者未必认为李克特量表各选项之间的间距相等 [26]。为解决这一问题，我们采用了 [26], 提出的 Snell 缩放程序，该程序使用最大似然估计，将李克特量表的序数类别置于连续尺度上的各个点，从而能够使用基于均值的参数方法。将该算法应用于我们的听力测试数据后，我们将七个李克特量表选项（强烈不同意、不同意、稍微不同意、中立、稍微同意、同意、强烈同意）映射为图4所示的数值。

评分结果与混合效应模型 我们使用线性混合效应模型（LMM） [17]检验生成类型对问卷问题结果的影响。选择 LMM 是因为它适用于重复测量，并且可以通过为每位参与者设置不同的随机截距来建模个体差异。我们将参与者的 MSI 分数 [19] 作为额外的自变量纳入模型，以评估音乐背景与评分之间的关系。最后，我们使用参与者 ID（PID）作为随机效应，以考虑不同参与者之间评分倾向的差异。混合效应模型通过 StatsModel 0.13.5 Python 库中的随机截距模型实现 [1]。在所有四个问题中，生成类型对评分结果均有显著影响，具体为 $p < .001$。MSI 分数对任何因变量均无显著影响，所有 $p \geq 0.22$。表4按问题列出了回归结果明细。

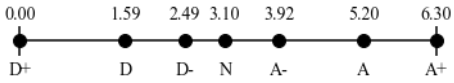

图 4. 将 7 点李克特量表值转换为连续量表（Snell 方法 [26]）。

表 4. 固定效应（系数、标准误和 p 值）与随机效应（PID 方差和标准误）的线性混合效应模型结果

	系数				**标准误**				$P > \|z\|$			
	Q1	**Q2**	**Q3**	**Q4**	**Q1**	**Q2**	**Q3**	**Q4**	**Q1**	**Q2**	**Q3**	**Q4**
截距	3.388	3.463	4.019	3.966	0.584	0.581	0.495	0.643	0.001	0.001	0.001	0.001
生成类型	-0.072	-0.897	-0.621	-0.565	0.072	0.082	0.070	0.069	0.001	0.001	0.001	0.001
MSI	0.009	0.005	0.006	0.008	0.007	0.007	0.006	0.008	0.220	0.446	0.315	0.310
PID方差	0.420	0.403	0.292	0.522	0.086	0.075	0.063	0.110	-	-	-	-

为直观展示不同生成类型之间的评分差异，我们使用原始李克特数据绘制了图5所示的李克特分布。除Q2：机器（“我喜欢这个循环中的音乐”）外，所有回应类别的中位数均为正值。对于李克特问题2—4，人工创作的片段的评分比机器生成的片段大约高出一个类别。对于Q1（“这个循环中的音乐具有原创性和音乐创意”），不同生成类型的中位数评分相同。Snell 缩放的 [26] 评分的均值、方差和标准差见表5。

音乐图灵测试结果 我们使用逻辑回归模型评估参与者区分人类创作与机器生成循环片段的能力。与李克特量表评分问题一样，模型将生成类型和 MSI分数作为自变量。为确保参与者的回答不会因熟悉音乐而偏向人类创作的循环片段，我们剔除了 83 个（共 1550 个）参与者表示认出了循环片段中音乐的数据点。我们使用 StatsModel Python库中的逻辑回归实现 [1] 计算模型结果。结果发现，生成类型对

感知到的生成来源（人类或机器）有显著影响，$p < 0.01$。MSI分数对感知到的生成来源没有显著影响，$p \geq 0.28$。为进一步了解参与者区分人类创作与机器生成循环片段的能力，我们绘制了表6中的混淆矩阵。尽管生成类型对感知到的生成来源有显著影响，但命中率仅比随机猜测高 10%。参与者正确判断生成类型的比例约为 60%，而人类和机器来源的分布相似。

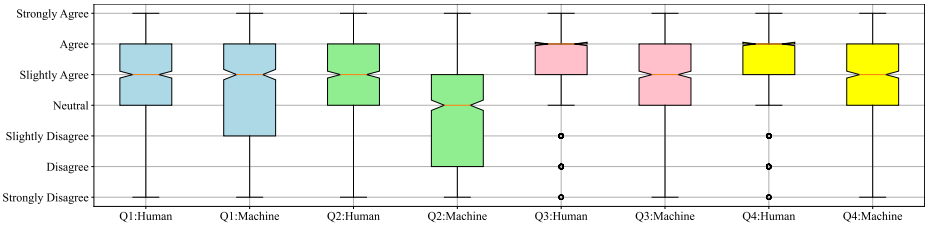

图5. 比较每个评估问题中，人类创作与机器生成循环片段的 7 点李克特评分分布。

表5. 李克特评分统计。H：人类；M：机器。

	中位数		Mean		方差		标准差	
	H	M	H	M	H	M		
Q1	稍微同意	稍微同意	4.09	3.38	2.24	2.62	1.50	1.62
Q2	稍微同意	中立	3.89	3.00	2.93	3.04	1.71	1.74
Q3	同意	稍微同意	4.50	3.88	1.83	2.55	1.35	1.60
Q4	同意	稍微同意	4.60	4.03	1.94	2.73	1.39	1.65

表6。 真实标签与预测生成类型的混淆矩阵。

	人类	机器
人类	0.6	0.4
机器	0.42	0.58

自由回答结果参与者可以在文本框中解释他们对听力测试中每个片段的评分。我们使用 WordCloud Python 库 [18] 生成词云，将每种生成类型回答中出现频率最高的 40 个词可视化。以下词语被用作停用词并从词云中移除，因为它们频繁出现但缺乏信息量：loop、melody、sounds 和 feels。如图6所示，与人类创作的片段相比，“simple”“boring”和“repetitive”更常用于描述机器生成的片段。另一方面，两种类型的片段中，“interesting”“smooth”和“coherent”的字号大致相同。“Good”是两类中都常见的词，但更常用于描述人类创作的片段。



图 6. 听力测试自由回答中机器生成（左）和人工创作（右）片段的基于频率的词云。

5 讨论

我们重构 DadaGP 数据集的方法总体上成功地促使模型生成可循环的输出。然而，Hard Repeats 和 Barred Repeats 数据集格式的输出质量存在出乎意料的差异。预训练硬重复模型平均生成的循环片段数量与基线模型相同，这表明该模型无法学会哪些特征构成可循环的乐句。从头开始训练时，Hard Repeats 模型的表现甚至更差，陷入局部最小值，所能做到的最好结果只是生成休止符和偶尔出现的长音符。模型似乎学会了专注于较容易处理的低密度音轨，却忽略了歌曲中高密度音轨的相关特征。

Barred Repeats 模型在推理期间生成循环片段方面表现出色，预训练模型和从头训练的模型平均生成的循环片段数量都是基线的 3 倍。尽管该模型在这一循环指标上表现优异，预训练 Barred Repeats 模型的验证损失却随时间严格递增，表明它对训练数据过拟合了。显然，模型学会了放置重复小节，却没有学会在有意义的位置放置它们。与预训练版本不同，从头训练的 Barred Repeats 模型确实收敛了，并生成了平均音符密度更高的循环片段，这表明它成功学会了创作具有音乐意义的循环片段。

我们的主观聆听测试使我们能够超越客观的音符密度指标，评估生成循环片段的质量。统计评估显示，生成类型对全部四项李克特量表评分问题均有显著影响，表明生成结果可以与人工创作的结果区分开来。不过，所有问题的评分中位数均为正值，这令人鼓舞，也表明我们的方法有望成为一种演出工具。

按问题拆分李克特量表评分后，值得注意的是，Q1 要求评估原创性和音乐性，其不同生成类型之间的评分最为相似。音乐性和原创性尤其体现人类特征，人工智能模型向来难以模仿，因此这一结果令人意外。一种可能是，参与者将出乎意料的音乐输出视为“原创”，从而抬高了机器生成片段的评分。Q2 询问参与者是否喜欢某个片段，无论是机器生成的片段还是人类创作的片段，其中位评分都是最低的。

尽管我们要求参与者关注乐谱，而非数字乐器的音质，但在回答此类偏好问题时，他们或许很难不受合成音色以及人类和机器生成片段演奏中缺乏人类表现力的影响，因而给出偏低的评分。如图5所示，Q3 和 Q4 分别询问音乐连贯性和循环点平滑度，两者的评分分布非常相似，这表明参与者可能难以区分音乐连贯性与循环点平滑度的含义。这在一定程度上可能是由部分片段中存在子循环造成的。例如，使用我们的提取算法时，一个四小节循环乐段实际上可能由重复的两小节循环构成。由于重复段的数量不一，参与者在回答 Q4 时可能难以将注意力集中在确切的循环点上。

音乐图灵测试数据表明，生成类型对感知到的生成来源有显著影响，但两种类型的结果仍接近随机水平。尽管图灵测试在评估算法作曲系统的音乐性方面是否有用仍存在争议 [3], ，但在我们的研究中，它是衡量 LooperGP 模仿训练数据可循环性能力的有效指标。换句话说，它能帮助了解 LooperGP 是否学会了在令人信服的位置插入重复小节。根据我们的结果，LooperGP 能够在一定程度上令人信服地模仿人类作曲家，但仍有改进空间。

听力测试的样本量仍相当有限（31 位参与者，每人对每种条件下的 25 个项目进行评估，因此每种条件下总计约有 775 条观测）。然而，对于固定效应而言，线性混合模型中的统计功效不一定会随着观测总样本量的增加而提高 [17]。

6 结论与未来工作

LooperGP 的设计初衷是用于现场编程，因此我们将在这一背景下讨论其潜力。参考文献 [6] 指出，要使生成过程适用于实时编程环境，它必须调用简便、适用于多种音乐情境、可修改，且所需时间有限。由于 LooperGP 专注于输出的可循环性，并且可以通过指定乐器编制、调号/拍号和小节长度来设定，因此能够适应各种音乐情境。我们的算法无需限定时间范围，因为该模型没有任何时间依赖性。LooperGP 与许多其他用于音乐生成的深度学习方法一样，其缺点是缺乏修改控制。循环生成后，无法通过算法对其进行调整。不过，如 [23] 所示，可以将生成的循环重新输入系统作为引导，从而生成其变体。最后，只需调用一个简洁的 Python 函数即可轻松执行 LooperGP 的推理，因为所需参数仅为调、时长和小节长度。

总体而言，与基线相比，LooperGP 能够提高高密度、可循环输出的出现频率。主观评价显示，人们总体上对这些循环片段反应积极，

但也指出了一些有待改进的关键方面，即减少简单性和重复性。LooperGP 具备生成过程在现场编程中发挥作用所需的诸多特征，并特别注重生成可控输出，以适应大多数音乐情境。LooperGP 的时长控制和可循环生成使其成为现场表演场景中的可行选择，不过仍需与现场编程者进一步评估。

为回应听力测试中关于机器生成循环“重复”且“无聊”的评论，未来的工作可以通过调整循环提取算法以识别并过滤掉子循环，从而提高所提取循环片段的平均复杂度。在训练方面，还可以进一步探索超参数空间，例如增加随机失活或减小层大小，以防止部分训练设置中出现的早期过拟合。

最后，未来工作中一个令人兴奋的方向是将 LooperGP 系统集成到 Tidal Cycles [16] 或 SuperCollider [15]等现场编程框架中。 [13]中介绍的用于 Python 与 SuperCollider 之间通信的 OSC管道，可用于对这一概念进行原型设计，而无需将模型导出到新的环境中。同样重要的是，应将 LooperGP 封装为一个易于使用的界面，并提供简洁的应用程序编程接口。理想情况下，表演者应能够从模型生成的一组候选选项中挑选一个循环片段，并自动将其移调至所需的调号。一旦 LooperGP 被封装进面向表演者的界面，就可以通过将其应用于音乐会并收集观众和表演者的反馈，真正评估其作为现场编程功能的潜力。

7 致谢

本研究部分得到了EPSRC UKRI人工智能与音乐博士培训中心的支持（资助编号：EP/S022694/1）。

参考文献

1. Statsmodels。https://github.com/statsmodels/statsmodels（2022年），访问日期：2022-08-14 2. Ackley, D.H., Hinton, G.E., Sejnowski, T.J.：玻尔兹曼机的学习算法。Cognitive Science，第147–169页（1985年）3. Ariza, C.：作为批评家的审问者：图灵测试与生成音乐的评估。Computer Music Journal **33**，48–70（2009年）4. Bishop, P.A., Herron, R.L.：李克特条目反应及其他顺序测量的使用与误用。Int. Journal of Exercise Science **8**，297–302（2015年）5. Briot, J.P., Hadjeres, G., Pachet, F.D.：音乐生成的深度学习技术，第1卷。Springer（2020年）6. Brown, A.R., Sorensen, A.：通过现场编程与生成音乐互动。Contemporary Music Review **28**，17–29（2009年）

7. Chandna, P., Ramires, A., Serra, X., G´omez, E.：由直观音乐参数条件控制的音乐循环合成。载于：ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing（ICASSP）。第3395–3399页。IEEE（2021年）8. Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q.V., Salakhutdinov, R.：Transformer-XL：超越固定长度上下文的注意力语言模型。arXiv预印本 arXiv:1901.02860（2019年）9. Hadjeres, G., Pachet, F., Nielsen, F.：DeepBach：一种可操控的巴赫众赞歌生成模型。International Conference on Machine Learning，第1362–1371页（2017年）10. Hsu, J.L., Liu, C.C., Chen, A.L.：发现音乐数据中的非平凡重复模式。IEEE Trans. on Multimedia **3**，311–325（9 2001年）11. Huang, Y.S., Yang, Y.H.：流行音乐Transformer：基于节拍的富有表现力的流行钢琴作品建模与生成。载于：Proceedings of the 28th ACM International Conference on Multimedia。第1180–1188页（2020年）12. Ji, S., Luo, J., Yang, X.：深度音乐生成综合综述：多层级表示、算法、评估与未来方向。arXiv预印本 arXiv:2011.06801（2020年）13. Lan, Q., Tørresen, J., Jensenius, A.R.：RaveForce：一种用于音乐的深度强化学习环境。载于：Proceedings of the smc conferences。第217–222页。Society for Sound and Music Computing（2019年）14. Magnusson, T.：Sonic Writing：材料、符号与信号铭写技术。Bloomsbury Publishing USA（2019年）15. McCartney, J.：SuperCollider：一种新的实时声音合成语言。Proceedings of the Int. Computer Music Conference，第257–258页（1996年）16. McLean, A., Wiggins, G.：Tidal——用于音乐现场编程的模式语言。载于：Proceedings of the 7th sound and music computing conference。第331–334页（2010年）17. Meteyard, L., Davies, R.A.：心理科学中线性混合效应模型的最佳实践指南。Journal of Memory and Language **112**，104092（2020年）18. Mueller, A.：Word Cloud。https://github.com/amueller/word_cloud（2022年），访问日期：2022-08-14 19. M¨ullensiefen, D., Gingras, B., Musil, J., Stewart, L.：非音乐家的音乐性：用于评估普通人群音乐素养的指数。PloS one **9**（2），e89642（2014年）20. Nilson, C.：现场编程实践。载于：Proceedings of the 7th international conference on New interfaces for musical expression。第112–117页（2007年）21. Ramires, A., Font, F., Bogdanov, D., Smith, J.B., Yang, Y.H., Ching, J., Chen, B.Y., Wu, Y.K., Wei-Han, H., Serra, X.：Freesound循环数据集与标注工具。arXiv预印本 arXiv:2008.11507（2020年）22. Sarmento, P., Kumar, A., Carr, C., Zukowski, Z., Barthet, M., Yang, Y.H.：DadaGP：用于序列模型的标记化GuitarPro歌曲数据集。载于：第22届国际音乐信息检索学会会议论文集（2021年）23. Shih, Y.J., Wu, S.L., Zalkow, F., Muller, M., Yang, Y.H.：主题Transformer：利用主题条件Transformer进行符号音乐生成。IEEE Trans. on Multimedia，第1–1页（3 2022年）24. Stewart, J., Lawson, S.：Cibo：自主Tidal循环表演者。载于：Proceedings of the Fourth International Conference on Live Coding。第353页（2019年）25. Sullivan, G.M., Artino, A.R.：分析与解释Likert型量表数据。Journal of Graduate Medical Education **5**，541–542（12 2013年）26. Wu, C.H.：将Likert量表数据转换为数值分数的实证研究。Applied Mathematical Sciences **1**，2851–2862（2007年）