EvoMusic user tailored generation

Laurence Bonat*, Davide Cavicchini*, Lorenzo Orsingher*, Ettore Saggiorato*

I. Introduction

In this report we present the work done for the BIO-Inspired Artificial Intelligence course project.

Our objective for this project is to use evolutionary algorithms to dynamically enhance the generation of music based on the user's taste from a discrete set of interactions. We believe that the approach we devised can be extended to other generation modalities, and allow for personalized generations without the need of expensive re-training or user effort.

This report presents the final framework inspired by evolutionary techniques seen during the course and promising trends in the community to achieve the stated goal.

II. Related Work

Generation of user-aligned content is a big challenge of current generative ai systems. Current solutions revolve around three main approaches:

- **Fine Tuning** [1]: trains the parameters of the generative model, or part of them, to satisfy some user preferences.
- **Prompt Tuning** [2]: uses learnable "soft prompts" to condition the generation of a frozen pre-trained model.
- **Prompt Engineering** [3]: the user rewrites the prompt in different ways until a satisfactory result is reached.

However, these options require either expensive training of the generative model or an active effort from the user to obtain a satisfactory result. To avoid some of these problems, the community moved their attention to automated search approaches such as evolutionary algorithms.

III. Problem Statement & approach

Our objective is to generate music that aligns with any users' musical tastes by leveraging their interactions with our system and dynamically adapting to them.

However, this is a vague statement. What does it mean to align with users' tastes? How is our system supposed to understand this? What strategy should we use to align the generations? These questions are fundamental for devising an intelligent objective function to optimize for and the subsequent strategies used to improve the generations. Our solution revolves around the use of four principal components:

- Music scorer based on user tastes.
- User tastes approximation with available feedback.
- Conditional Music Generators.
- Evolutionary Strategies.

To join these components into the final system we use the schema illustrated in Figure 1. The evolution of the music will thus keep repeating the following steps in order:

1) Evolve music using the current understood user tastes
2) Generate the final playlist for the user
3) Get feedback from the user
4) Fine-tune beliefs on user tastes

These steps use the components we identified and put them into a functioning system whose overall objective is to evolve music in alignment with the user's interactions.

To start, we will examine the fitness function, which allows us to optimize our solutions. Here, we will briefly see what model we used, how we trained it, and how we use it. This will cover both the music score and the approximation from the feedback. Then we will take a look at the strategies we explored to evolve our solutions, comprehensive of the conditional generator used and how we manipulated it. Finally, we will arrive at our conclusion and results and try to give a brief intuitive understanding of what worked and why, as well as possible ways to improve our work.

IV. Fitness function - Music Scorer

Measuring the quality of a song is a challenging task as it cannot be easily quantified by a metric, and it's highly subjective to the listener. Here comes the need for an approximation of both the user preferences and the song characteristics. For the latter, it's enough to leverage a pre-trained music encoder, the resulting embedding can be used to compare the alignment with a similar representation of the user's tastes. User embeddings are not directly available and need to be approximated. Moreover, direct human feedback is both expensive and time-consuming. To overcome these limitations we developed an alignment model, namely *AlignerV2*, that projects the user embeddings, synthesized from a large music-interactions dataset, and music embeddings in a shared latent space, which allows us to calculate the similarity between the two embeddings and use it as a fitness function for the downstream tasks.

A. Synthetic users - AlignerV2 Model

The starting point was to generate synthetic users using a model inspired by [4]. Our synthetic users are based on the Last.fm dataset [5] [6] which contains the listening history of 992 unique users and 3 million interactions. To speed up the training we pre-extracted the song embeddings using the MERT encoder [7] and averaged the embeddings for all the frames of each song in order to obtain a tensor invariant to the length of the track. We then trained our model *AlignerV2*[IV-A] to project the song and user embeddings in a shared latent space, the architecture is shown in Figure 2(a). The MERT representation from 13 of its layers are weighted via a learned gating mechanism. For user embeddings, we use a simple feedforward network with skip connections and GELU activation functions. Our loss function of choice is an implementation of the InfoNCE loss [8] with learnable

temperature. To aid the contrastive learning and obtain a diverse set of positive samples (user-music interactions) we set up the dataset so that single users could appear multiple times in the same batch coupled with different songs they had listened to.

B. Synthetic Users Interactions

To unify real and synthetic users under a common interface, we needed an approach that does not rely on direct access to the real user embeddings.

A natural idea is to map the fitness function results to a variety of human-like interactions, such as like, dislike, hated, ... However, translating a continuous probability distribution into such a broad discrete space would introduce unnecessary complexity. Fortunately, IV-A accurately learns the probability of a user listening to a song, reflecting their preferences. For this reason, we adopted a more straightforward approach: if a user listens to a song, they like it; if not, they don't.

To achieve this, we leveraged the fitness function domain $[-1, 1]$, to easily map negative values to -1 and positive values to 1. By mapping the output space to two discrete values, we achieved a computationally efficient representation of user behavior that remains both interpretable and believable, as the feedback corresponds to actions users could realistically take.

C. Users Approximation

To estimate the user embeddings based on user interactions and song data, enabling the computation of a fitness function to generate improved songs. We leverage a variant, softmax-based, InfoNCE. Additionally, we tested various initialization methods for user embeddings, *mean*, *random*, and *mean+random noise*, finding the mean initialization to be the most effective.
Fine-tuning of the model supports two modes:

- **Classic**: Iterates multiple times over the same data, allowing for repeated fitting.
- **Minibatch**: Employs a circular memory that updates with each fine-tuning step, utilizing randomly shuffled minibatches. This method yielded the best results.

Evaluation of approximated synthetic users is conducted on the test portion of the static dataset used for training the model in section IV-A, aligned with the current reference user.

D. Noise penalty

Additionally, a noise penalty is added to the fitness function to filter out noisy solutions. The techniques applied to the audio are the following:

1) Clipping detection
2) DC-offset
3) High-frequency content/harsh noise detection
4) Spectral flux for discontinuity
5) Prolonged Near-Silence detection

E. Final Fitness Formulation

Let $f(\cdot)$ be the user embedding encoder, $g(\cdot)$ the music embedding encoder, $h(\cdot)$ the noise penalty estimator, and u and m the user and music embedding respectively, the final fitness function can be written as:

$$\mathcal{F} = \frac{f(u) \cdot g(m)}{||f(u)|| \cdot ||g(m)||} + h(m) * noise\_weight \quad (1)$$

V. Music Evolution

In this section, we examine the model used to generate music conditioned on the solutions from the evolution process and discuss the exploration strategy we explored.

To generate diverse music, we considered GANs and Encoder-Decoder models, as explored in [9]. However, for greater flexibility and interpretability, we chose Meta's MusicGen, which generates music from text descriptions [10].

This choice enabled us to explore two distinct approaches: evolving text descriptions directly or manipulating the T5 encoder's embeddings as a latent representation. The genetic coding for the latter is detailed in Section V-B.

A. Prompt Optimization

Integrating Large Language Models (LLMs) provides significant advantages by facilitating dynamic and adaptive search and being able to work on a higher level of abstraction, making them ideal for tasks lacking a clear exploration space. Recent research, such as [11], demonstrates the effectiveness of LLM-driven optimization in enhancing evolutionary search strategies. Additionally, their text-based nature aligns well with our music generation pipeline, which utilizes structured prompts for creative exploration.

The explored strategies share three key common parameters that govern the evolutionary process:

- Roulette-wheel selection (*sample*): Introduces stochasticity by enabling probabilistic selection of individuals based on their fitness, fostering diversity and preventing premature convergence.
- Novelty Injection (*novel_prompts*): Defines the fraction of the population generated from scratch in each iteration, ensuring a continuous influx of novel solutions.
- Elite Retention (*elites*): Specifies the proportion of the population carried over unchanged from one generation to the next, safeguarding the most promising candidates.

*1) **Full LLM**:* In the full LLM strategy, generation is entirely LLM-driven. The key innovation lies in the LLM's role in not just mindlessly creating music prompts but selecting and evolving through its internal reasoning.

Initially, a set of pre-generated prompts is used to generate songs. Each song is then evaluated based on our fitness criteria, evolution proceeds by selecting the best prompts, preserving a subset through elite selection, re-initialize the novel prompt, and filling the remaining population with what the LLM proposes after receiving the current population results and its internal reasoning. Crucially, the LLM is not limited to text generation; it also performs critical analysis. A meta-prompt instructs the model to reflect on why certain solutions resulted

in lower scores and to generate improved alternatives in the next iteration.

Rather than merely producing static outputs, the system actively participates in optimizing the prompts for the songs, without needing a pre-defined optimization strategy.

*2) **LLM Evolve**:* Using more biologically inspired techniques, we can try to emulate genetic algorithms using a similar approach to [12], where an LLM is used to apply the genetic operators directly to the text prompts. This way, we can leverage more intuitive domain-specific operators and task the LLM to target specific properties of the textual prompts, such as the music genre and instruments used.

Our implementation allows the definition of a sequence of genetic operators applied to the original population to generate the next one. These operators are applied sequentially with the specified probability until the whole population is filled. To get the first samples to start the genetic process a tournament selection is used, where $\mathcal{N}$ prompts are sampled with a uniform distribution, and from those we either directly take the best ones, or use the roulette wheel selection. A pseudo-code implementation of this algorithm can be seen in 1.

B. Embedding Optimization

A more traditional approach is numerical optimization, which allows us to leverage already existing and mature algorithms. These approaches, require a genome that encodes for the properties of the children, and from which we can compute the fitness values.

In our case, we can use embeddings (*token embeddings*) from the embedding matrix of the T5 encoder. The token embeddings are used to obtain the genetic code by flattening them along the first dimension.

On top of these embeddings, we applied standard numerical optimization strategies and chose to test CMA-ES, SNES, and genetic algorithms.

*1) **CMAES - SNES**:* These numerical optimization techniques work on distributions by generating the offspring using specific statistical measures to search for an optimization path in the latent space. CMAES [13] works by adapting the new samples and population through a multivariate Gaussian distribution. It uses a covariance matrix to adapt the σ and the Covariance Matrix C to generate a successful offspring. SNES [14] updates a population of candidates through a multivariate Gaussian distribution but uses a diagonal covariance matrix, assuming full independence between dimensions.

*2) **Genetic Algorithm**:* Genetic algorithms are a class of optimization algorithms inspired by the process of natural selection. They operate on a population of candidate solutions, each represented by a genome encoding the solution's properties. The algorithm iteratively evolves the population by applying genetic operators such as selection, crossover, and mutation to generate new offspring. Genetic algorithms are well-suited for optimization tasks where solutions are encoded as vectors of parameters and the relative position of these parameters affects the solution's quality. This is precisely our case, as our genome directly maps to the token sequence used to generate the music.

VI. Experimental Results

In this section, we will look at the experimental results we were able to obtain for our pipeline. We will try to validate our approach, while also discussing the problems we encountered during the evolution process. This section will also try to highlight open issues in our current implementation on which further work and experimentation might be beneficial. To compare the different strategies, we test them using the following minimal settings: optimize for the dynamic approximation of the user 0, the user 0 itself, and a piece of music as control to see if the generation evolves the same feeling and genre.

A. Users

First of all, we need to validate what arguably is the foundation of our work, the modeling of user tastes. This module guides the whole evolution process, directing the solution toward the best music for the user. The analysis can be split into two components: the user modeling, and the user approximation.

To model the users' behavior we trained the aligner model showcased in section IV-B for 150 epochs with a batch size of 256 and a learning rate of 0.001, with a scheduler to reduce the learning on plateau. For validation, we use the ROC AUC score, which can tell us how well the model can distinguish between positive and negative samples. The results of the training are shown in figure 2(b), the final score obtained by the model on the test set is 0.73, which is a valid result considering the complexity of the task.

To validate the user approximation we use IV-A to generate a synthetic dataset of interactions. These are then used to sample a selection of 100 positive and negative samples, from which we are able, in approximately 10 epochs, to achieve a cosine similarity greater than 0.85 with respect to the scores obtained directly from IV-A. From this, we can conclude that this module works as intended.

B. Prompt Optimization

In this section, we will take a look at the experimental setups and results for direct text optimization using population-based approaches. Unfortunately, we are unable to test state-of-the-art models, as we lack both computational resources and OpenAI credits. For this reason, our tests only employ models such as llama3.2 up to 3B versions, deepseek-r1:1.5B, and GPT-4o-mini. Unsurprisingly, the best-performing model is GPT-4o-mini, however, we still encountered problems with generating well-formatted results even with this model. The system prompt used for our LLM-based experiments is available in the Appendix as Prompt 1.

*1) **Full LLM**:* To test this strategy, we developed three prompts 4, 5, and 6, with the latter yielding the best results. Local models like `ollama-r1:1.5B` suffered from prompt stagnation, limiting novelty and reducing performance compared to larger models.

Disabling sampling produced decent results, as shown in Plot 3, while enabling sampling led to inconsistent results, even with larger populations and models. Adjusting novelty

rate or elite selection worsened performance, and reducing them below 0.1 (default) also had negative effects, as summarized in Table II.

Despite these issues, the generated music was always structurally valid. However, in the poorly performing test on `music 0` (with sampling enabled), the output had the wrong feel and genre.

*2) **LLM Evolve**:* For this strategy, we tested multiple configurations of parameters. Intuitively, this being a genetic algorithm approach to the problem an important parameter to tune is the *Population Size*, in here we will present results obtained using a population of 100 individuals which we found to be a good balance between computational time and performances. Another important parameter to tune is the *Tournament Size* used for parent selection, which was set to 25. This might seem a very high number, however, we found that using this value in conjunction with the sampling strategy explained in section V-A, yielded a good trade-off. Additionally, the sampling strategy, being applied also to the elitism, allowed to sometimes veer off from the best solutions (local minima) and find even better solutions.

Regarding the operators used, this approach to our problem can be considered one of the most expensive, as it requires calling the LLM model multiple times for each new individual. For this reason, we also had to trade off the expressivity of the genetic process by computing multiple children from the same parents. The final operators used are showcased in the appendix 2 and 3, which try to both exploit the knowledge from the parents (using the crossover operation) and explore new solutions (using mutation). At each generation, we also inject novel prompts into the population to avoid collapse.

The obtained results are summarized in Table I. The final results confirm the validity of this approach, a qualitative analysis of the control run on the music also confirms that the generated prompts do indeed follow the correct feel of the target music. Additionally, we can see from the progress plots of the relevant metrics in Figure 6, that the metrics follow the typical curve of genetic algorithms: in the early phase, it's easy to make progress, while in the mid and late phases, the progress becomes much slower and sometimes stagnates in local optimum solutions.

C. Embedding Optimization

*1) **CMAES - SNES**:* CMAES and SNES have quite similar-looking results. For this reason, runs with the SNES algorithm are not included since they result in similar behaviors to CMAES runs.

The difficulty with this modality is that the space we are exploring is strongly sparse, and presents many strong correlations between dimensions. This can be seen in Figure 8 obtained using T-SNE projection which tries to maintain the relations between cluster elements. In the resulting projection, clusters of different genres are not connected, making the exploration of such space challenging.

The table IV shows the results obtained on different modalities and runs with the CMAES algorithm. In Figure 7 we can see that the run made in the music mode plateaus after a small number of generations. The other runs show a better trend but after the 25th generation, they all tend to plateau. The music generated with this strategy has poor quality, with much of it being almost noise.

*2) **Genetic Algorithm**:* Similarly to the LLM Evolve approach, we tested multiple configurations of population and tournament size. Preliminary experiments showed that crossover and mutation rates did not have as much impact on the final results, compared to the parameters mentioned before, so we decided to keep them fixed at 0.5 and 0.1 respectively, focusing the analysis on the more relevant experiments.

Table III shows the results across the different modalities, higher population sizes seem to have a positive impact on the final results, at the cost of a much longer computation time since the largest bottleneck for this strategy is the generation of the music samples.

While from the raw numbers, it might seem that the tournament size did not have a decisive impact on the final results, taking a closer look at the curves tells a different story: higher tournaments can converge faster, sometimes to good solutions, but they also tend to stagnate in local optima. Figure 4 clearly shows how the enhanced exploration of the smaller tournament size allows for finding better solutions in the long run.

Another parameter that heavily influenced the final results was the presence of elitism. No elitism causes unstable evolutions and the inability to converge to a good solution, this effect is clearly shown in the graph in Figure 5. On the other hand, excessive elitism led to stagnation, similar to the effect of a larger tournament size. A balanced elitism rate of 0.1 provided stable and steady convergence toward high-quality solutions.

VII. Conclusions

To conclude this report, let's summarize the general results for the different modules and strategies.

While prompt-based optimization showed promising results, with the **LLM Evolve** strategy being the most effective with small models, we believe there is a lot of margin for improvement in the full LLM strategy by simply testing with more reasoning-capable models such as `o1`. However, these approaches suffer from the generation methods of these LLM systems and results could highly vary from run to run.

On the other hand, token embedding optimization resulted in mixed performance, with CMA-ES being unable to perform above random performance; while the genetic algorithm approach showed solid and predictable results.

VIII. Contributions

Davide worked on the part of evolution, Lorenzo on the synthetic users, Ettore on the user approximation and Laurence on the inference of the models. Experiments on CMA-ES and SNES were performed by Laurence, on genetic algorithms by Lorenzo, Full LLM strategy by Ettore, and LLM Evolve strategy by Davide.

REFERENCES

[1] M. A. Bakker, M. J. Chadwick, H. R. Sheahan, M. H. Tessler, L. Campbell-Gillingham, J. Balaguer, N. McAleese, A. Glaese, J. Aslanides, M. M. Botvinick, and C. Summerfield, "Fine-tuning language models to find agreement among humans with diverse preferences," 2022. [Online]. Available: https://arxiv.org/abs/2211.15006

[2] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," 2021. [Online]. Available: https://arxiv.org/abs/2104.08691

[3] J. Oppenlaender, "A taxonomy of prompt modifiers for text-to-image generation," *Behaviour amp; Information Technology*, vol. 43, no. 15, p. 3763–3776, Nov. 2023. [Online]. Available: http://dx.doi.org/10.1080/0144929X.2023.2286532

[4] J. Lee, K. Lee, J. Park, J. Park, and J. Nam, "Deep Content-User Embedding Model for Music Recommendation," 2018. [Online]. Available: https://arxiv.org/abs/1807.06786

[5] P. Global. Last.fm. [Online]. Available: https://www.last.fm/

[6] O. Celma, *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[7] Y. Li, R. Yuan, G. Zhang, Y. Ma, X. Chen, H. Yin, C. Xiao, C. Lin, A. Ragni, E. Benetos, N. Gyenge, R. B. Dannenberg, R. Liu, W. Chen, G. Xia, Y. Shi, W. Huang, Z. Wang, Y. Guo, and J. Fu, "MERT: acoustic music understanding model with large-scale self-supervised training," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=w3YZ9MSlBu

[8] K. Sohn, "Improved deep metric learning with multi-class n-pair loss objective," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf

[9] N. SiddarthaReddy, M. SaiPrakash, V. Varun, V. Vaddina, and S. Gopalakrishnan, "Leveraging latent evolutionary optimization for targeted molecule generation," *2024 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:271310269

[10] J. Copet, F. Kreuk, I. Gat, T. Remez, D. Kant, G. Synnaeve, Y. Adi, and A. D'efossez, "Simple and controllable music generation," *ArXiv*, vol. abs/2306.05284, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259108357

[11] M. Wong, Y.-S. Ong, A. Gupta, K. K. Bali, and C. Chen, "Prompt evolution for generative ai: A classifier-guided approach," in *2023 IEEE Conference on Artificial Intelligence (CAI)*. IEEE, Jun. 2023, p. 226–229. [Online]. Available: http://dx.doi.org/10.1109/CAI54212.2023.00105

[12] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang, "Connecting large language models with evolutionary algorithms yields powerful prompt optimizers," 2024. [Online]. Available: https://arxiv.org/abs/2309.08532

[13] N. Hansen and A. Ostermeier, "Completely Derandomized Self-Adaptation in Evolution Strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001. [Online]. Available: https://ieeexplore.ieee.org/document/6790628

[14] T. Schaul, T. Glasmachers, and J. Schmidhuber, "High dimensions and heavy tails for natural evolution strategies," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 845–852. [Online]. Available: https://dl.acm.org/doi/10.1145/2001576.2001692

IX. Appendix

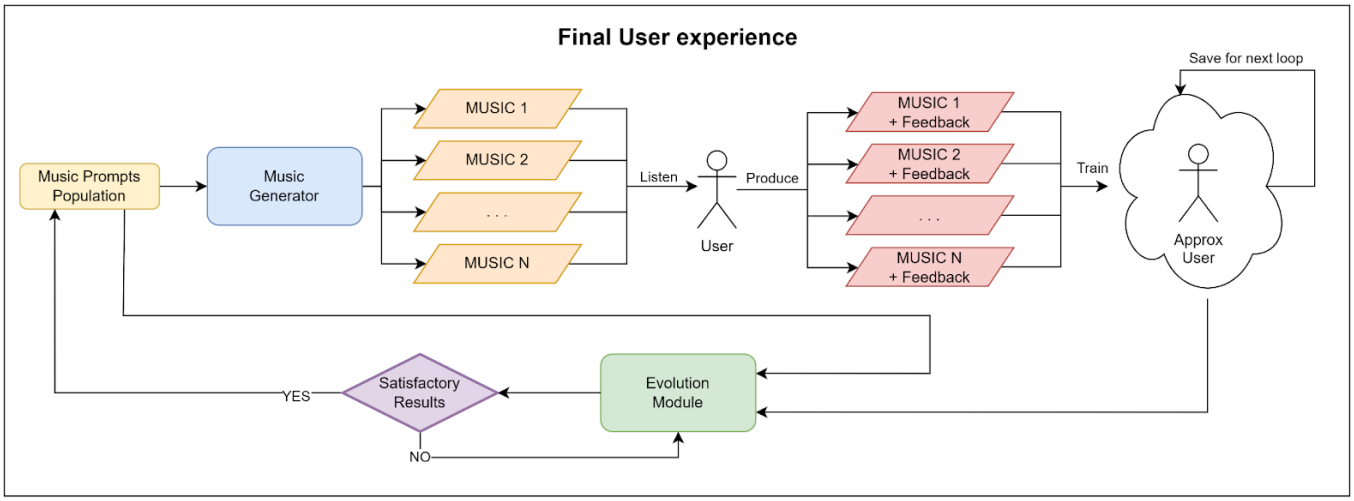
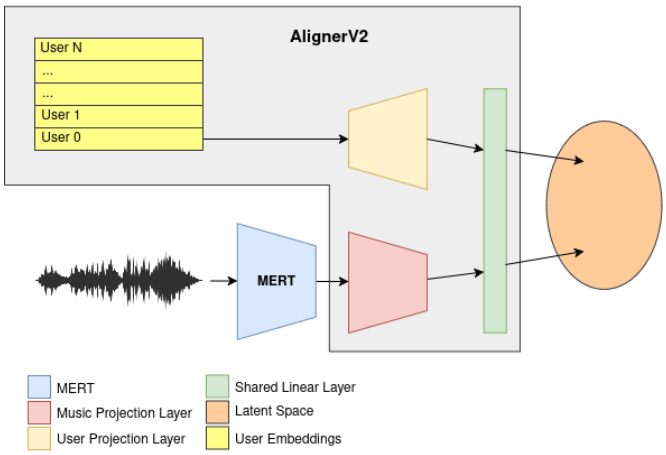
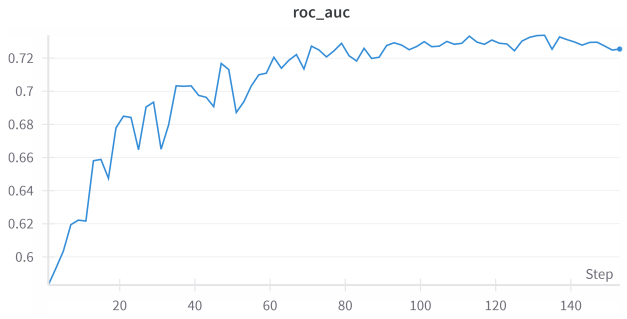


Fig. 1: Pipeline of the EvoMusic architecture



(a) AlignerV2 architecture



(b) Validation ROC AUC for the training of *AlignerV2*

Fig. 2: Overview of the *AlignerV2* architecture and its validation ROC AUC.

System Prompt

Prompt 1: System prompt used to obtain valid and parsable output from the LLM models.

You produce prompts used to generate music following the requests of the user. You should always respond with the requested prompts by encasing each one in <prompt> and </prompt> tags XML style. DO NOT USE THEM ANYWHERE ELSE. Examples of valid output prompts are:
1. <prompt> Text of prompt. </prompt>
2. <prompt> Another prompt. </prompt>

Algorithm 1 Pseudo-Code implementation of the LLM Evolve strategy

Set the generation counter, $t = 0$ initialize the strategy parameters.
OPs is the list of operators, and $p(op)$ is the probability for that operator.
$I(0)$ is the input size for the first operator, $O(op)$ is the output size for the operator op.
Create and initialize the population, $C(0)$, of μ individuals

while stopping condition not true **do**
  for each individual, $x_i(t) \in C(t)$ **do**
    Evaluate the fitness, $f(x_i(t))$;
  end for

  NewPrompts $\leftarrow$ [ ]
  NewPrompts $+=$ Sampled/Top Elites
  NewPrompts $+=$ Novel Prompts

  while len(NewPrompts) $< \mu$ **do**
    Tournament $\leftarrow$ sample $\mathcal{N}$ from $C(t)$
    Input $\leftarrow$ Sample/Top $I(0)$ from Tournament

    for op in OPs **do**
      if Uniform$[0, 1) < p(op)$ **then**
        Out $\leftarrow$ LLM(op, Input)
      else
        Out $\leftarrow$ sample $O(op)$ from Input
      end if
    end for

    NewPrompts $+=$ Out
  end while

  $C(t + 1) =$ NewPrompts
  $t = t + 1$
end while

Cross-Over Operator

Prompt 2: Prompt for the crossover operation. The probability of this operator is 1 with 2 parents and 1 child.

You have two prompts. Cross-pollinate them by combining their distinctive elements (genres, instruments, overall mood).
Create one new prompt that seamlessly blends the unique traits of each parent. Do not exceed the length of the parent prompts, and in general contain it to a single phrase.
Original prompt(s):
{prompts}

Mutation Operator

Prompt 3: Prompt for the mutation operation. The probability of this operator is 1 with 1 parent and 3 children.

Introduce a small random change to the original prompt — such as swapping out one instrument, altering the genre slightly, or tweaking the emotional tone. Generate three new prompts with a subtle modification.
Keep the new prompts contained into a single phrase, with length not exceeding the parent.
Original prompt(s):
{prompts}

Full LLM - Basic Prompt

Prompt 4: *Generate {num_generate} music prompts for generating music based on the classification and scores of the previous prompts. You should balance exploration and exploitation to maximize the score. BEFORE giving the music prompts, you should spend time to reason on the classification and scores of the previous prompts, and understand what makes a prompt successful for the user, what makes it fail, how to combine the acquired knowledge and where we are not exploring, for example if a music generne is not being explored or if the prompts are too similar.You should also try to understand and reason about the user preferences based on the scores and the classification of the prompts, and how to exploit this knowledge to generate better prompts. AFTER this careful reasoning about the current evaluation, you should generate a diverse set of prompts that are likely to be successful tying to not repeat te same patterns and content in the requested format.Here is the current population with their similarity scores and ranking for the current generation:{ranking} after the reasonin, generate only the next generation of prompts with a population of {num_generate} prompts.*

Full LLM - Improved Prompt

Prompt 5: *Your task is to generate a population of* ***{num_generate}*** *diverse and effective music prompts. Each prompt should be designed with a balance of exploration and exploitation based on the evaluation (similarity scores) fromprevious prompts.### Before Generating Prompts:-* ***Analyze Previous Prompts****: Categorize each prompt into 'Successful' or 'Failed' based on their similarity score on the current generation: {ranking}.- Successful: High score, led to creative or engaging results.- Failed: Low score, resulted in repetitive or uninteresting outputs.-* ***Understand Success Factors****: Identify why successful prompts were effective. Consider aspects like genre specificity, emotional appeal, and uniqueness.-* ***Identify Areas for Improvement****: Note which music genres are underrepresented or if prompts lack diversity.-* ***User Preferences****: Deduce user preferences from the scores. For instance, higher scores might indicate a preference for certain emotional tones or styles.### After Analysis:-* ***Generate New Prompts****: Create a diverse set of* ***{num_generate}*** *prompts that avoid repetition. Ensure each prompt is unique and taps into unexplored areas.-* ***Format****: Present each prompt clearly, using numbered lists with bold headings (e.g., **Genre Focus**, **Emotional Appeal**).-* ***Clarity and Specificity****: Each prompt should be concise yet specific enough to guide the model effectively.### Example Prompts:1. **Genre Focus:** Create a haunting melody in theKeys ofreason's jazz improvisation style.2. **Emotional Appeal:** Craft an uplifting track using strings and synths for a motivational vibe."*

Full LLM - Further Improved Prompt

Prompt 6: *Current ranking: {ranking}. Analyze previous prompts and categorize each as Successful (>50) or Failed (<=50) based on their scores (scores are normalized in range [0,100]). Identify success factors such as genre specificity, emotional appeal, and uniqueness. Determine areas for improvement, such as underrepresented genres or lack of diversity. Deduce user preferences from scores, recognizing favored emotional tones or styles. Then, generate {num_generate} new, high-quality music prompts that balance exploration and exploitation to maximize the total score. Treat the total score as the reward you want to maximize. Ensure diversity, avoid repetition, and explore unexplored areas. Format each prompt clearly in a numbered list with bold headings (e.g., Genre Focus, Emotional Appeal) while maintaining clarity and specificity. Example Prompts: 1. Genre Focus: Create a haunting melody in theKeys ofreason's jazz improvisation style. 2. Emotional Appeal: Craft an uplifting track using strings and synths for a motivational vibe.*

mode	best fitness	final max fitness	mean fitness	median fitness
dynamic 0 (wrt true usr emb)	0.740	0.680	0.274	0.273
dynamic 0 (wrt estimated usr emb)	0.738	0.728	-0.090	-0.169
user 0	0.718	0.703	0.333	0.324
user 500	0.607	0.607	-0.207	-0.295
music	0.660	0.655	0.598	0.604

TABLE I: Fitness comparison for different objectives for LLM evolve optimization strategy.

mode	model	best fitness	final max fitness	mean fitness	median fitness	user cosine	prompt
dynamic 0 (no sampling)	gpt-4o-mini	0.68	0.68	-0.04	-0.01	0.72	6
dynamic 0 (sampling)	gpt-4o-mini	0.48	0.48	-0.09	-0.19	0.58	5
static 0 (sampling)	gpt-4o-mini	0.66	0.66	0.18	0.17	-	5
music 0 (sampling)	gpt-4o-mini	0.62	0.66	0.55	-	-	5
r1 0 (no sample)*	r1-1.5b	-	0.79	0.32	-	-	5
r1 0 (no sample)*	r1-1.5b	-	0.62	0.24	-	-	4

TABLE II: Full LLM - Fitness comparison for different objectives. *Short runs (3 epochs, 25 population size and best songs).

mode	population	tournament size	best fitness	mean fitness	best fitness app	mean fitness app
dynamic	25	4	0,362	0,239	0,112	-0,102
dynamic	100	8	0,511	0,075	0,526	0,015
dynamic	100	25	0,497	0,251	0,281	-0,006
user 0	25	4	0,585	0,375	-	-
user 0	100	8	0,687	0,428	-	-
user 0	100	25	0,687	0,455	-	-
music	25	4	0,631	0,587	-	-
music	100	8	0,638	0,604	-	-
music	100	25	0,641	0,599	-	-

TABLE III: Fitness comparison for different objectives for GA optimization strategy. For the dynamic mode experiments it's also included the reference scores calculated on the target user.

mode	best fitness	mean fitness
dynamic 0	0.598	0.174
user 0	0.745	0.638
user 500	0.526	-0.152
music	0.682	0.604

TABLE IV: Fitness comparison for different objectives for CMAES (8 Epochs, population 100, tournament size 25, stdev$_init$30)

Fig. 3: Full LLM - Best long-run - dynamic 0 (no sampling)II - It's clear that the song generation is improving overtime and the median fitness is getting better; the validation cosine scores wrt the reference user grows overtime, indicating that the framework is correctly working.

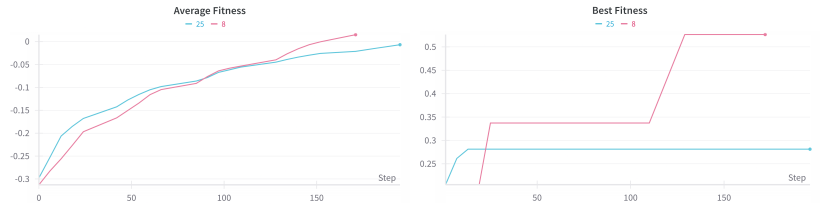


Fig. 4: A comparison of two GA runs with a population of 100 solutions. Although the average fitness trends upward for both configurations, the smaller tournament size and hence boosted exploration might help the population to escape local minima and reach better solutions.

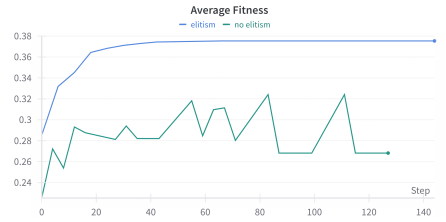


Fig. 5: The positive effects of elitism on the GA evolution.

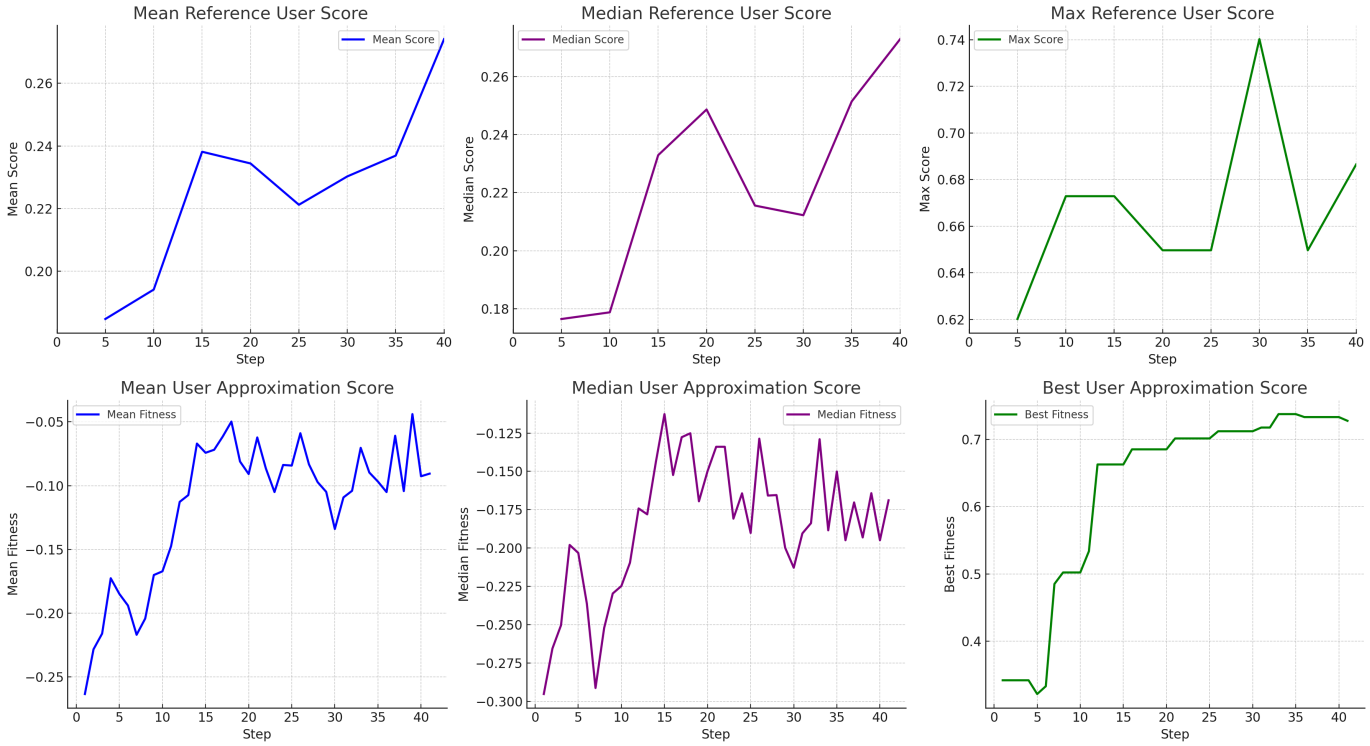


Fig. 6: LLM Evolve fitness function mean, median a best of population. (top) is wrt the reference user embedding. (bottom) is wrt the approximated user embedding. These trends clearly show the optimization curves typical of genetic algorithms.

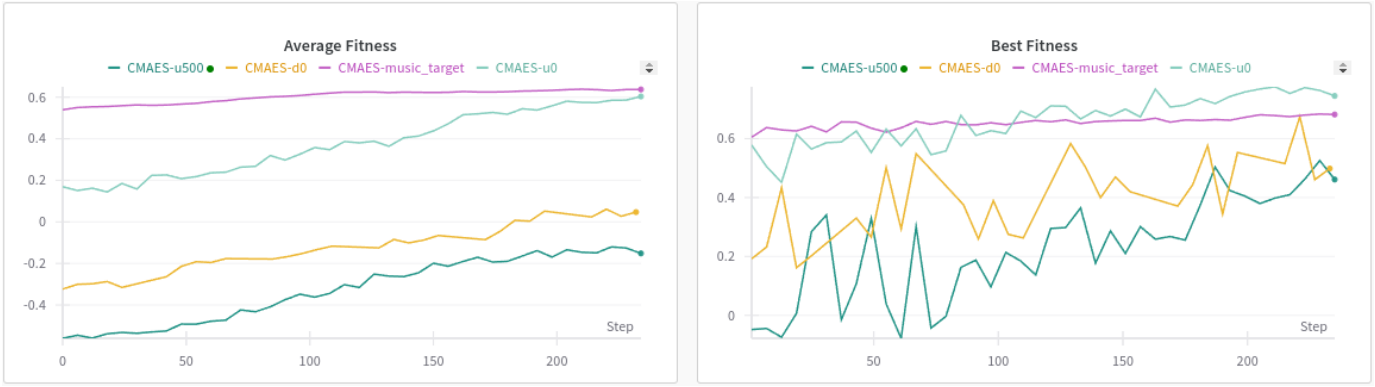


Fig. 7: A comparison between different runs made in dynamic mode(d0), music mode, user mode on user 0(u0) and user 500(u500)

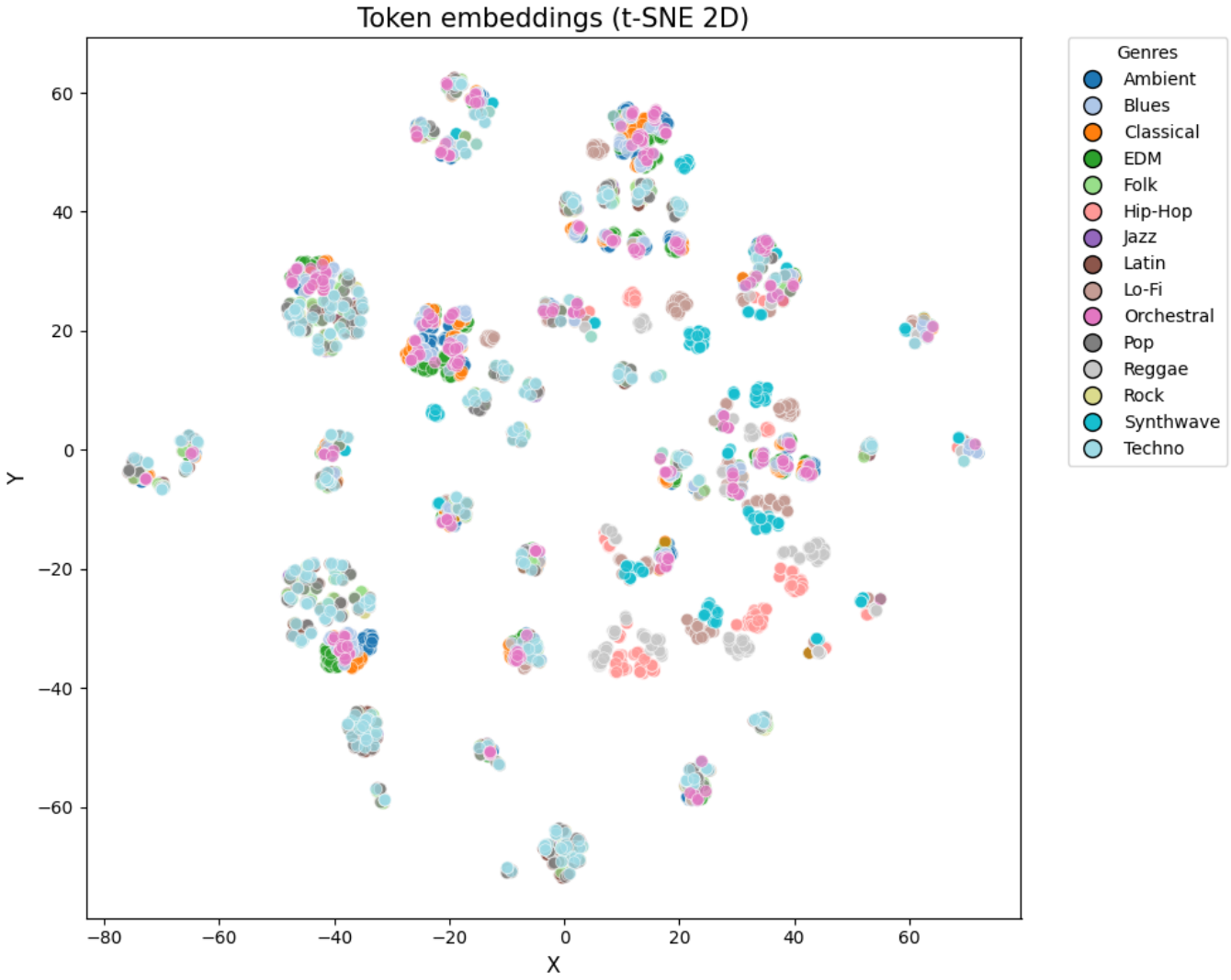


Fig. 8: t-SNE representation of the token embeddings latent space