Matcha-TTS

Mathis Lecry MATHIS.LECRY@ENSAM.EU
Master Systèmes Avancés et Robotique
Sorbonne Université et Arts et Métiers
Paris, France

Paul-Marie Demars NOM.PRENOM2@MAIL.FR
Master Systèmes Avancés et Robotique
Sorbonne Université et Arts et Métiers
Paris, France

Minh MATHIS.LECRY@ENSAM.EU
Master Systèmes Avancés
Sorbonne Université
Paris, France

Yucheng MATHIS.LECRY@ENSAM.EU
Master Systèmes Avancés et Robotique
Sorbonne Université et Arts et Métiers
Paris, France

Editor: Machine Learning Avancé (2025-2026)

Abstract

Le résumé synthétise en environ 200 mots la tâche abordées, les problèmes associés, la solution proposée, et les résultats principaux

This project presents the re-implementation of the Matcha-TTS model, a fast and probabilistic text-to-speech architecture based on Optimal-Transport Conditional Flow Matching (OT-CFM)(**?**). The objective was to reproduce the results of the paper by Mehta et al. (2024) by implementing the architecture from scratch. We implemented a Transformer-based text encoder, a duration predictor, and a 1D U-Net decoder driven by an Ordinary Differential Equation (ODE). The model was trained on the LJSpeech dataset. This report details our architectural approach, the implementation choices made to separate the mathematical logic from the neural network, and the results obtained during training.

Keywords: Synthèse vocale, Flow Matching, U-Net, Deep Learning, Reproduction.

1. Introduction

L'introduction présente le contexte général lié à la tâche, description de la tâche à traiter, détail des problèmes qui y sont associés.

Text-to-Speech (TTS) synthesis has witnessed many advancements in recent years, shifting from rigid concatenative methods to neural generative models capable of producing near-human natural speech. Currently, TTS algorithms are dominated by Diffusion Probabilistic Models (DPMs), which model speech generation as an iterative denoising process (3). However, these models suffer from a major drawback : their slow inference speed. Indeed, generating a high-quality spectrogram requires a large number of sampling steps to solve the underlying Stochastic Differential Equations (SDEs), creating a difficult trade-off between synthesis speed and audio quality (3; 4).

In this context, our project focuses on the study and implementation of **Matcha-TTS**, a new architecture proposed by Mehta et al. (2024) aiming to resolve this latency issue without sacrificing quality (3). The primary task of this model is to convert input text into a Mel-spectrogram (subsequently converted into a waveform by a vocoder), utilizing an innovative approach called Optimal-Transport **Conditional Flow Matching (OT-CFM)** (1). Unlike classical diffusion methods based on score matching, Flow Matching learns a vector field that defines simpler and more direct probabilistic trajectories (straight lines) between the noise distribution and the data distribution (speech) (**?**).

Implementing this solution presents several technical challenges and associated problems that we have addressed :

1. **Computational Efficiency :** Designing a decoder based on Ordinary Differential Equations (ODEs) capable of generating high-quality results in fewer steps compared to models like Grad-TTS (2).
2. **Alignment and Duration Modeling :** The model must jointly learn to align text with audio and predict phoneme durations, necessitating the integration of a monotonic alignment module and a robust duration predictor (3).
3. **Encoder Architecture :** To improve the modeling of long-term dependencies in text, the architecture integrates modern attention mechanisms, specifically **Rotary Positional Embeddings (RoPE)**, which allow for better relative position extrapolation compared to classical sinusoidal embeddings (**?**).

This report details our approach to reproducing this non-autoregressive and probabilistic architecture, analyzing how shifting from SDEs to ODEs via Flow Matching enables the creation of a TTS system that is fast, lightweight, and high-performing.

2. Présentation de l'algorithme

Une présentation synthétique de la solution ré-implémentée de l'article. Précisez et justifiez les éventuelles différences avec l'article de référence

Dans le cadre de ce projet, nous avons entrepris la ré-implémentation complète de l'architecture **Matcha-TTS** présentée par Mehta et al. (2024) (3). L'objectif était de reproduire les résultats de l'état de l'art tout en proposant une base de code plus accessible et modulaire, adaptée à un contexte pédagogique et expérimental. Cette section détaille notre approche et justifie les écarts volontaires par rapport à l'implémentation officielle.

2.1 Architecture Globale et Flux de Données

Notre solution conserve l'architecture fondamentale non-autorégressive basée sur le *Optimal-Transport Conditional Flow Matching* (OT-CFM). Le pipeline de génération suit les étapes suivantes :

1. **Encodage du Texte :** Une séquence de phonèmes est transformée en représentations latentes par un encodeur Transformer.
2. **Alignement et Durée :** Un module d'alignement monotone (MAS) aligne ces représentations avec le spectrogramme cible durant l'entraînement. Un prédicteur de durée apprend simultanément à estimer la longueur de chaque phonème pour l'inférence.
3. **Décodage par Flow Matching :** Un décodeur U-Net 1D, conditionné par le texte et le temps t, prédit le champ de vecteurs nécessaire pour transformer un bruit gaussien en spectrogramme Mel via la résolution d'une ODE (Ordinary Differential Equation).

Cette structure respecte fidèlement la méthodologie décrite dans l'article original (3) et utilise les principes théoriques du Flow Matching (1).

2.2 Différences avec l'Article de Référence et Justifications

Bien que le cœur mathématique soit identique, nous avons opéré plusieurs simplifications et refactorisations par rapport au code officiel (`shivammehta25/matcha-tts`). Ces choix sont motivés par une volonté de clarté et de maîtrise du code.

2.2.1 Simplification de la Configuration (Abandon de Hydra)

L'implémentation officielle repose lourdement sur **Hydra** pour la gestion des hyperparamètres, avec une structure de fichiers de configuration YAML complexe et imbriquée (plus de 30 fichiers de config détectés dans le dépôt original).

Notre approche : Nous avons remplacé ce système par une configuration explicite et directe en Python.

— **Justification :** L'utilisation de Hydra, bien que puissante pour des expérimentations à grande échelle, obscurcit la lecture des paramètres pour une ré-implémentation. En définissant les hyperparamètres (canaux cachés, taux d'apprentissage, etc.) directement dans le code ou via des arguments simples, nous rendons l'architecture plus transparente et plus facile à déboguer pour notre équipe.

2.2.2 Refactorisation du Flow Matching (Séparation des Responsabilités)

Dans le code original, la logique mathématique du Flow Matching est souvent entremêlée avec la définition des modèles PyTorch.

Notre approche : Nous avons isolé la logique du Flow Matching dans un module dédié `matcha/models/components/flow_matching.py`. Ce fichier contient spécifiquement les classes pour l'interpolation OT-CFM et le calcul de la perte, indépendamment de l'architecture du réseau de neurones.

— **Justification :** Cette séparation respecte le principe de responsabilité unique. Elle permet de modifier la méthode de génération (par exemple, changer le solveur d'ODE ou le type de bruit) sans toucher à l'architecture du décodeur U-Net, facilitant ainsi la maintenance et la compréhension des équations différentielles sous-jacentes (**?**).

2.2.3 Gestion des Données et Phonémisation

L'article original propose un pipeline de pré-traitement très générique capable de gérer de multiples datasets et langages via des scripts complexes (`matcha/text/cleaners.py`, `utils/generate_data_statistics.py`).

Notre approche : Nous nous sommes concentrés spécifiquement sur le dataset **LJSpeech** (anglais, mono-locuteur). Nous avons réécrit un `DataModule` simplifié (`ljspeech_datamodule.py`) et un `Dataset` (`ljspeechDataset.py`) qui intègrent directement la phonémisation et la conversion en spectrogrammes Mel.

— **Justification :** En restreignant le périmètre au dataset LJSpeech, nous avons éliminé une grande quantité de code "boilerplate" nécessaire à la gestion multi-locuteur et multi-langue, ce qui nous a permis de nous concentrer sur la performance du modèle acoustique lui-même (**?**).

2.2.4 Optimisation et Logging

Le code de référence inclut des outils de logging avancés (WandB, Neptune, etc.) et des callbacks PyTorch Lightning complexes pour la visualisation.

Notre approche : Nous avons implémenté un système de logging plus léger, utilisant principalement les fonctionnalités natives de PyTorch Lightning pour le suivi des pertes (*loss*) et la sauvegarde des *checkpoints*.

— **Justification :** Pour notre échelle d'entraînement (sur une seule GPU RTX A6000), un monitoring simplifié est suffisant et réduit les dépendances externes, rendant le projet plus facile à installer et à exécuter sur différents environnements.

2.3 Problématique et Contribution

Les auteurs de Matcha-TTS cherchent à résoudre le compromis vitesse/qualité des modèles TTS actuels. Ils introduisent une architecture encodeur-décodeur non-autorégressive. Contrairement à des modèles comme Grad-TTS qui utilisent la diffusion basée sur le score (SDE), Matcha-TTS utilise le Flow Matching (ODE), ce qui simplifie la trajectoire de génération et réduit le nombre de pas nécessaires (NFE - Number of Function Evaluations).

2.4 Architecture du Modèle

Le modèle se compose de trois blocs principaux que nous avons ré-implémentés :

1. **Encodeur de Texte** : Un réseau Transformer qui convertit la séquence de phonèmes en une représentation latente et prédit la durée de chaque phonème pour l'alignement.
2. **Alignement Monotone (MAS)** : Un algorithme qui génère un chemin optimal entre le texte et le spectrogramme audio cible.
3. **Décodeur (Flow Matching)** : Un U-Net 1D qui prend en entrée un spectrogramme bruité, le temps t, et le texte aligné conditionnel μ, pour prédire le champ de vecteurs $v_t(x)$ permettant de reconstruire le son.

3. Données

Présentation des bases de données utilisées. Précisez et justifiez les éventuelles différences avec l'article de référence.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

4. Evaluation expérimentale

4.1 Description de l'expérience

Description de l'expérience réalisée, méthodologie et métriques d'évaluation.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

4.2 Résultats

Présentation des résultats expérimentaux obtenus et comparaison par rapport à ceux de l'article de réference.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

4.3 Discussion

Discussion critique à partir des résultats obtenus

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

5. Conclusion

Conclusion : un résumé synthétique des principaux résultat obtenus et présentation des principales pistes d'amélioration possibles.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Références

[1] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[2] Shivam Mehta. Matcha-tts repository. `https://github.com/shivammehta25/Matcha-TTS`, 2024.

[3] Shivam Mehta, Ruibo Tu, Jonas Beskow, Éva Székely, and Gustav Eje Henter. Matcha-tts: A fast tts architecture with conditional flow matching. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024.

[4] Vadim Popov, Ivan Vovk, Vladimir Gogoryan, Tasnima Tashev, and Mikhail Kudinov. Grad-tts: A diffusion probabilistic model for text-to-speech. In *International Conference on Machine Learning*, pages 8599–8608. PMLR, 2021.