Présentation:

Timewarp est une méthode qui combine un flow normalisant inversible et un échantillonnage MCMC pour accélérer l'exploration de l'espace conformationnel moléculaire.
Sur Flow normalisant (Normalising Flow), il est un réseau neuronal inversible, qui apprend à transformer une distribution simple (comme une gaussienne, bruit aléatoire) en une distribution complexe, typique des conformations moléculaires.
Pendant la phase d'apprentissage, on dispose de paires de structures issues des dynamiques moléculaires (état t et état t+τ). On utilise l'**inverse du flow** pour projeter chaque configuration future (x(t+τ)) dans l'espace latent gaussien, **conditionné sur l'état initial**.
Lors du sampling, on procède dans le sens direct : on **échantillonne des variables gaussiennes**, puis on les passe à travers le flow (conditionné sur l'état initial) pour générer de nouvelles structures réalistes.
->En résumé, le flow permet de proposer des « sauts » efficaces et adaptés à la physique du système.
C'est donc le flow qui nous fournit des candidats réalistes pour explorer de nouveaux états conformationnels.

Sur l'Echantillonnage MCMC, Chaîne de Markov Monte Carlo est une méthode où chaque nouvelle proposition de structure dépend exclusivement de l'état précédent , bien sûr, c'est le principe de la chaîne de Markov. À chaque étape, on utilise le flow pour proposer une nouvelle structure, mais cette proposition doit être acceptée selon un critère statistique qui s'appelle la règle de Metropolis-Hastings, qui prend en compte l'énergie et la probabilité. Cela permet de garantir que l'exploration reste fidèle à la distribution physique correcte -> la distribution de Boltamann.

Le pipeline général de mon projet se compose de 4 étapes principales. Je vais vous présenter en détail comment j'ai réalisé chaque étape, en explicitant à la fois les principes, les formules clés, ainsi que la logique de chaque partie. Enfin je vais expliquer comment j'ai analysé et interprété des résultats obtenus.

Première étape, Données d'entraînement:
Cette formule résume le principe du modèle: (解释公式，为什么有这些param)
L'entrée, ce sont des conformations moléculaires, c'est-à-dire les positions de tous les atomes d'un molécule. Pour chaque paire d'apprentissage, on utilise la structure à un instant t et la structure à l'instant t+tau, les deux configurations consécutives sur la trajectoire MD.
À chaque configuration, on ajoute une variable de vitesse, échantillonnée selon une loi normale centrée -> un bruit gaussien. En plus, on introduit aussi des

variables latentes gaussiennes, elle aussi tirées d'une loi normale standard. Ces deux types de bruit gaussien ont **un rôle complémentaire**:
1) Les vitesses gaussiennes correspondent à la physique réelle — dans une simulation MD à l'équilibre, les vitesses suivant naturellement une distribution normale, donc on respecte la réalité physique et on évite le surapprentissage.

2) Les variables auxiliaires permettent au modèle d'explorer toutes les possibilités de perturbations thermiques sur la structure — c'est un moyen d'augmenter la diversité des exemples et d 'apprendre la distribution complète des évolutions possibles.
Grâce à ces deux sources de bruit, le modèle apprend non seulement la dynamique moyenne, mais aussi toutes les transitions possibles sous l'effet du bruit thermique. Et puis, il devient possible de prédire la prochaine conformation probable d'une molécule à partir d'une structure initiale et d'un bruit

C'est pourquoi on prépare les datasets de cette façon :
Pour couvrir la diversité des évolutions possibles, il nous faut des paires de structures moléculaires successives issues des trajectoires MD, chaque paire étant enrichie de vitesses auxiliaires tirées de N(0, I).
Trois étapes :
– Générer plusieurs trajectoires MD
– Échantillonner des paires (x^p(t), x^p(t+τ)) à intervalle régulier
– Ajouter des vitesses auxiliaires N(0, I) à chaque paire
La Figure 1 montre que la distribution des vitesses auxiliaires générées est parfaitement conforme à la loi normale standard attendue, validant la méthode de génération et l'adéquation physique du dataset.

Deuxième étape, Architecture générale:

L'architecture de Timewarp combine trois briques principales :
– des couches de couplage RealNVP
– des modules Atom Transformer
– une attention par noyau multi-têtes (kernel self-attention)

Tout d'abord, au niveau le plus externe, on retrouve les Coupling Layers de RealNVP, qui effectuent des transformations réversibles sur les coordonnées et les vitesses de la molécule.
A l'intérieur de chaque Coupling Layer, les paramètres de transformation — scale et shift — sont calculés par des modules Atom Transformer.
Chaque Atom Transformer, enfin, repose sur une attention locale basée sur la distance entre deux atomes, grâce au module de kernel self-attention multi-têtes.
Cette architecture permet de modéliser efficacement la dynamique moléculaire tout en respectant les contraintes physiques intrinsèques du système.

-> Kernel self-attention:

Un noyau RBF est une fonction qui mesure la similarité entre deux atomes en fonction de leur distance, plus les points sont proches, plus la similarité est forte, plus l'influence est forte.
Dans une molécule, les interactions physiques sont locales: les atomes voisins s'influencent fortement, alors que les atomes éloignés ont une influence négligeable. Pour modéliser cette localité, on définit l'attention non pas par la similarité des vecteurs caractéristiques (comme dot-plot dans transformers), mais directement en fonction de la distance euclidienne entre chaque paire d'atomes.

Le poids w_ij mesure l'influence de l'atome j sur i. L'analogie avec la régression à noyau gaussien est parfaite : on fait une moyenne pondérée, pondération décroissante avec la distance. Ici, Le paramètre l contrôle la portée spatiale (lengthscale).
Une fois les poids w_ij calculés, on met à jour la représentation de chaque atome i comme une somme pondérée des caractéristiques de tous les autres atomes j, ici V est une matrice linéaire arpprise. Les voisins proches comptent davantage dans cette moyenne.

Pour chaque tête, on utilise un lengthscale différent pour calculer les poids d'attention via la formule 2, puis on met à jour les caractéristiques selon la formule 3. Le mécanisme multi-head consiste simplement à répéter ce processus en parallèle avec plusieurs valeurs de lengthscale, ce qui permet d'agréger l'information à différentes échelles spatiales.

-> Atom Transformer:
Dans l'Atom Transformer, pour calculer les poids d'attention, on utilise la position x^p(t) des atome, et non vitesses ou d'autres variables. Cela respecte la contrainte physique: dans une molécule, les interactions locales à un instant donné ne dépendent que de la configuration spatiale.

Pour chaque atome, on concatène la position, l'embedding de type atomique, et une variable latente (selon le contexte).
En entrée, les vecteurs concaténés passent d'abord par un MLP (phi_in), qui les projette dans un espace de dimension supérieure.
Puis, ils traversent plusieurs couches Transformer empilées : kernel self-attention, normalisation, connexions résiduelles, MLP.
Ce bloc agrège dynamiquement l'information des voisins proches selon la distance.
Enfin, chaque atome a une nouvelle représentation riche, qui passe dans un MLP de sortie (phi_out) pour fournir un vecteur 3D (paramètre de scale ou shift pour le flow).
Le rôle de l'Atom Transformer : fournir dynamiquement les paramètres des transformations du flow, et non générer directement une distribution.

-> RealNVP Coupling Flow:

À chaque couche du flow, on applique :
– D'abord on met à jour la position à l'aide de la vitesse actuelle, puis
– On met à jour la vitesse à l'aide de la nouvelle position.
Les paramètres scale et shift sont calculés par les Atom Transformers, assurant cohérence physique et réversibilité.

Ici, scale contrôle l'étirement local, shift contrôle la translation locale.
Au lieu de prédire brutalement la position finale, le modèle applique une succession de petites déformations, ce qui reflète les petites variations typiques de la dynamique moléculaire.

En empilant toutes ces couches, le modèle transforme deux bruits gaussiens indépendants—un pour la position, un pour la vitesse—en une configuration moléculaire complète à l'instant t+τ, c'est-à-dire : les nouvelles positions et vitesses (x^p(t+\tau), x^v(t+\tau)).
Ce n'est donc pas une prédiction directe : le modèle apprend à reconstituer toute l'évolution de la dynamique moléculaire, en tenant compte des petites variations successives.

Dans la **Phase I - Entraînement**, notre objectif est d'apprendre la distribution conditionnelle des évolutions futures, en utilisant le flow **dans le sens inverse** :
À l'entraînement, pour chaque paire de structures, on part de la conformation cible (future), on la ramène dans l'espace latent gaussien à l'aide du flow inverse.
Cette projection permet au modèle d'apprendre la distribution complète des évolutions futures, sous une forme abstraite et universelle, directement exploitable lors de la génération.

Donc, pour expliquer **comment** on apprend concrètement cette distribution conditionnelle, il faut d'abord comprendre ce que signifie chaque terme de la formule (6).
Je vais la décomposer en plusieurs parties pour que ce soit plus clair:
1) x(t+tau): la future configuration moléculaire que l'on veut prédire, pour une condition initiale donnée.
2) f^-1: C'est la projection inverse par le flow : on prend la structure future réelle et on la ramène dans l'espace latent gaussien, conditionné sur l'état initial.
3) N}(.; 0, I): C'est la densité de proba dans l'espace gaussien: C'est la densité de probabilité dans l'espace gaussien : plus une structure projetée colle à cette distribution, plus elle est probable pour le modèle.
4) det: C'est le jacobien de la transformation : il ajuste la probabilité pour prendre en compte la déformation de l'espace par le flow. En pratique, on maximise la vraisemblance des trajectoires futures observées :On entraîne le modèle à donner u**ne forte probabilité à toutes les transitions futures effectivement observées**, pour chaque état initial — c'est-à-dire à bien couvrir

la diversité des évolutions possibles, pas seulement la plus probable.

公室问题什么是k：
Dans la formule, l'indice k correspond à chaque paire de structures (x(t), x(t+τ)) de notre dataset. Pour chaque paire, le modèle prédit la probabilité d'observer l'état futur, sachant l'état initial. Ensuite, on fait simplement la moyenne du log-likelihood sur tous les échantillons du dataset. Cela revient à optimiser la capacité globale du modèle à reproduire la distribution des trajectoires futures.
Mais **d'où vient cette diversité de futurs possibles** ?
Lors de l'apprentissage, on inverse le processus dans RealNVP : on part de la structure future observée, et on la projette dans l'espace latent gaussien, couche par couche, en suivant exactement la même architecture. Cela permet d'optimiser tous les paramètres du modèle pour qu'il puisse ensuite générer (en sampling) toute la diversité des trajectoires futures plausibles à partir d'un même état initial.

训练结果：
J'ai bien implémenté et entraîné le modèle Timewarp sur mon propre jeu de données (~1500 paires).
Pour entraîner le modèle, on maximise la vraisemblance des états futurs observés — ce qui correspond mathématiquement à maximiser le log-likelihood sur le dataset (formule (7)). En pratique, cela revient à **minimiser la loss**, qui est simplement le négatif du log-likelihood. Voilà pourquoi, sur nos courbes d'entraînement, on voit la loss baisser et la log-vraisemblance augmenter de manière très corrélée.
On constate que la perte d'entraînement et de test converge très bien, sans gap significatif : cela montre que le modèle apprend parfaitement la structure des exemples vus. Mais, avec un dataset aussi petit (~1500 paires), il y a probablement un phénomène de sur-adaptation aux exemples précis, ce qui expliquerait la courbe aussi lisse.
En regardant l'erreur de position, on observe également une diminution nette. Cependant, pour valider la **généralisation** du modèle, il serait nécessaire de tester sur d'autres types de molécules ou d'augmenter la diversité du dataset.

Phase II - Exploration : MCMC avec le flow entraîné
Après avoir entraîné le modèle (phase I), on fige les poids du flow : il devient alors un générateur de propositions dans un algorithme d'exploration stochastique — le MCMC de Metropolis-Hastings.

- **À chaque itération :**

1. À partir de la structure courante, je génère une nouvelle proposition de conformation via le flow (ce qui est beaucoup plus efficace qu'un simple bruit aléatoire).
2. J'applique la règle d'acceptation Metropolis-Hastings (équation 10), qui prend en compte à la fois l'énergie (Boltzmann) et la probabilité donnée par le modèle.
3. Après chaque proposition, je ré-échantillonne complètement les

variables auxiliaires (ici, les vitesses) par un pas de Gibbs sampling : on les remplace par un nouveau tirage selon une loi normale standard (équation 11).

Ce schéma permet de garantir l'exploration correcte de l'espace des états, conformément à la théorie (voir équations 8-11). En répétant ce cycle de propositions/acceptations/ré-échantillonnages, on finit par obtenir des échantillons qui suivent réellement la distribution Boltzmann cible.

公式解释：
La distribution cible ici est la distribution de Boltzmann, qui favorise les conformations de faible énergie. Ainsi, plus l'énergie potentielle d'une conformation est basse, plus elle a de chances d'être acceptée dans l'algo.
Le p_theta représente notre modèle flow entraîné. Pour chaque conformation actuelle X^p, il génère une nouvelle proposition de conformation, basée sur ce qu'il a appris à partir des trajectoires MD. Les paramètre theta (params dans le modèle)sont fixé après entraînement.
Le tilde sur Xm indique qu'il s'agit de la proposition générée par le modèle, à partir de l'état courant X_m. Ce n'est pas encore accepté — on décide après selon la probabilité d'acceptation.
L'équation (10) donne la probabilité d'acceptation : c'est le minimum entre 1 et le ratio du produit de la probabilité cible (distribution de Boltzmann) et de la probabilité du modèle flow, calculés pour la proposition et pour l'état courant. Plus ce ratio est élevé, plus on a de chances d'accepter la proposition. Si le ratio est inférieur à 1, on accepte avec cette probabilité — sinon, on accepte toujours.

exploration结果:
Après mes résultats pour ça, le Ramachandran plot est un diagramme de dispersion qui représente la distribution es angles dièdres phi et psi de la chaîne principale des protéines. Il permet de visualiser la diversité conformationnelle accessible à la molécule.
Ici, on l'utilise pour comparer les états générés par le modèle (en rouge) avec les conformations issues de la trajectoire MD (en bleu), afin d'évaluer la capacité du modèle à reproduire la distribution physique réelle.

On observe que, même en augmentant le nombre d'échantillons MCMC de 500 à 500 000, la différence sur la distribution des angles dièdres reste relativement limitée. Cela ne s'explique pas seulement par le nombre d'itérations, mais surtout par la **taille restreinte du jeu de données d'entraînement** : On justement utilise 1500 paires (x(t), x(t+τ)).

Enfin, Pour valider le réalisme des structures générées, j'ai extrait la 101^{e} frame de la trajectoire d'échantillonnage, et on observe que la structure reste parfaitement cohérente avec la physique du système. Cela confirme que le modèle ne produit pas de structures aberrantes durant l'exploration.