# Rapport de Projet:
# Inférence de modèles d'interactions à l'aide de transformers

Elian MANGIN, Hadrien CRASSOUS
Centrale-Supélec, Université Paris-Saclay

Encadrés par:
Pascale LE GALL (MICS), Boutheina BANNOUR (CEA List),
et Céline HUDELOT (MICS)

Octobre 2024 - Avril 2025

## Contents

# 1 Contexte du projet

Les interactions sont des modèles décrivant les flux de communication entre des acteurs, dans des systèmes concurrents ou des systèmes distribués. De nombreux systèmes réels sont naturellement décrits par des interactions:

- les protocoles de transactions [1–6].
- l'envoi de messages au sein d'une flotte de véhicules [7]
- ou encore la communication entre sous-systèmes d'une application [8–10]

Les interactions permettent de formaliser les spécifications d'un système en décrivant précisément les échanges entre composants, tout en offrant une représentation synthétique et visuelle de ces échanges complexes. Cependant les applications ne sont pas toujours développées avec leurs modèles associés, et la plupart des systèmes sont maintenus sans avoir accès au modèle d'interaction. Sans disposer du modèle d'interaction, il est plus difficile d'étudier le comportement du système. Il y a alors un grand enjeu à automatiser la génération du modèle d'interaction.

Pour cela, il faut collecter des suites de messages issus de l'interaction, appelées **traces**, représentant le comportement à l'éxecution du modèle, et **inférer** l'interaction.

Notre projet consiste à implémenter des méthodes d'apprentissage profond pour inférer les interactions à partir de traces.

# 2 Reformulation du problème

Une **trace** est une séquence d'actions qui est obtenue en observant l'éxecution d'un modèle d'interaction, c'est-à-dire que la succession d'actions suit les règles décrites par l'interaction. En collectant de nombreuses traces ou des traces très longues, on peut espérer avoir assez d'information pour inférer l'interaction. Cependant, il n'existe pas d'algorithme pour inférer les interactions à partir des traces, et il est difficile de déterminer quelle serait la quantité minimale de traces ou d'information pour écrire un tel algorithme.

Dans la continuité du travail de nos encadrants et de Kazuma Ikesaka, nous approchons le problème comme un problème de traduction séquence-séquence. Pour cela, on utilise les Transformers [11–13], la technique d'apprentissage profond qui a révolutionné les méthodes de traduction multi-langues. Nous souhaitons entraîner ces modèles de traduction avec des données synthétiques d'interactions et évaluer leur performance sur des données réelles.

# 3 Objectifs du projet

Dans le cadre de ce travail de recherche nous souhaitons:

- Définir des méthodes de génération de données synthétiques et d'évaluation des modèles d'inférence.
- Entrainer des modèles d'inférence des interactions à partir des traces, et valider les performance par des données réelles.
- Etudier les facteurs limitant l'apprentissage (taille des interactions, taille de trace, taille de la base de données)

# 4 Méthodologie

Nous avons eu, tout au long du projet, des points réguliers avec nos encadrants, à un rythme d'environ toutes les deux semaines. Ces réunions nous permettaient de faire un état des lieux de notre avancement, de discuter des difficultés rencontrées, et de définir les prochaines étapes. D'un point à l'autre, cela nous laissait le temps de développer et de tester les idées discutées. Par ailleurs, nous avons été invité à travailler une journée au CEA avec nos encadrants, accompagnés par Erwan MAHE et Kazuma IKESAKA.

La répartition des tâches s'est faite naturellement au fil du projet. Beaucoup des fonctionnalités ont été dévelopées conjointement et discutées entre nous deux. Certains aspects ont été traités spécifiquement par un élève: en général, Elian s'occupait de l'implémentation des algorithmes d'entraînement et du lancement des entraînements et Hadrien était plus focalisé sur la mise en place des métriques d'évaluation et l'exploitation des exemples réels.

Concernant la génération de données, nous avons travaillé conjointement sur cette partie, en alternant entre l'exploration d'approches synthétiques et l'utilisation de données réelles transformées. Cela a demandé de nombreuses itérations, afin d'obtenir des données variées et pertinentes pour notre tâche.

Nous avons également adopté une démarche itérative : à chaque cycle, nous testions les résultats, évalué les performances, puis ajustions notre approche en fonction des observations. Cette méthodologie nous a permis de rester flexibles et de nous adapter rapidement aux obstacles techniques ou méthodologiques rencontrés.

# 5 Schéma d'ensemble

Ce rapport décrit notre implémentation et nos expériences réalisées au cours du projet. Notre implémentation contient:

- Un module de génération de données synthétiques 7.1
- Un module d'apprentissage profond implémentant un transformer 7.2
- Une base de données d'interactions provenant de la littérature 7.6
- Un module d'évaluation des modèles entrainés 7.7

Nos encadrants nous ont également sollicité pour contribuer à un article sur le sujet du projet, dont la rédaction vient de commencer.

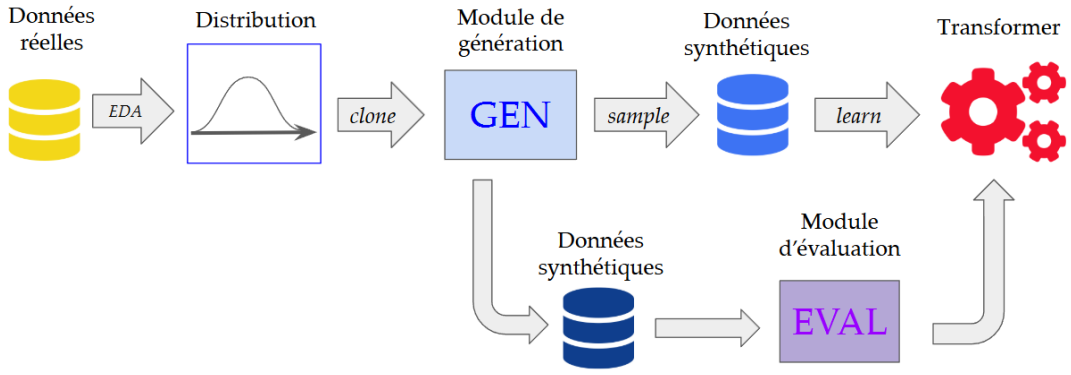



Figure 1: Schéma d'ensemble de notre implémentation

# 6 Etat de l'art

## 6.1 Le langage des interactions

Pour manipuler les interactions, nous utilisons le formalisme du langage des interactions. Il permet de construire des interactions avec des actions (émission ou réception d'un message entre des acteurs appellés lifelines) et des opérateurs entre les interactions.

### 6.1.1 Diagramme de séquence

Le principal avantage des interactions est leur représentation graphique facile à lire, représentées en **Diagrammes de Séquence**. Décrivons leurs éléments principaux à l'aide du diagramme de séquence illustré dans la figure 2

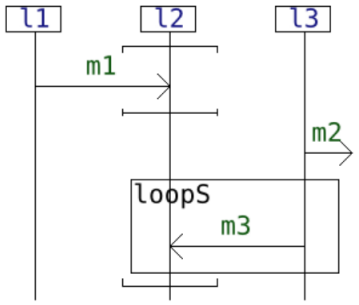



Figure 2: Diagramme de séquence basique [1]

Chaque acteur est associé à une ligne verticale appelée **ligne de vie** (ici $l_1$, $l_2$ et $l_3$). Les comportements sont décrits comme des **traces**, c'est-à-dire des successions d'actions qui consistent soit en l'émission, soit en la réception d'un message par une ligne de vie. Dans la représentation diagrammatique :

- **L'émission** d'un message est représentée par une flèche sortant d'une ligne de vie.
- **La réception** est indiquée par une flèche entrant dans une ligne de vie.

Les échanges de messages (une émission suivie d'une réception correspondante) sont représentés par des flèches horizontales contiguës reliant deux lignes de vie. Le nom du message émis ou reçu est indiqué au-dessus de la flèche correspondante (par exemple $m_1$, $m_2$ et $m_3$ dans cet exemple).

### 6.1.2 Contraintes et ordonnancement

Étant donné que les actions sont discrètes et atomiques, les échanges de messages imposent un **ordre causal** : l'émission d'un message doit toujours précéder sa réception. Dans l'exemple donné, $l_1$ doit émettre $m_1$ avant que $l_2$ puisse le recevoir.

Des **opérateurs de haut niveau** permettent de structurer les actions de communication atomiques en scénarios complexes : *Séquençage, Composition parallèle, Choix, Répétition*

Ces opérateurs peuvent être utilisés pour définir comment les actions sont programmées. Une particularité des interactions réside dans la distinction entre le **séquençage strict** et le **séquençage faible** :

- Le séquençage strict impose un ordre entre les actions, quelles que soient leurs lignes de vie respectives.
- Le séquençage faible impose un ordre uniquement entre les actions se produisant sur une même ligne de vie.

### 6.1.3 Représentation graphique

La plupart des opérateurs sont représentés par des boîtes annotées (par exemple, `loopS` désigne une boucle strictement séquentielle dans l'exemple). Cependant :

- L'opérateur de séquençage faible correspond à la direction verticale (haut vers bas) du diagramme.
- Les régions concurrentes sont représentées par des crochets sur des lignes de vie spécifiques (par exemple, sur $l_2$ dans l'exemple).

### 6.1.4 Exemple d'utilisation

Dans l'exemple donné 2 :

- Le séquençage faible force $l_3$ à émettre $m_2$ avant d'émettre une instance de $m_3$, car ces deux actions se produisent sur la même ligne de vie ($l_3$).
- En revanche, $m_1$ et $m_2$ peuvent être émis dans n'importe quel ordre, car ils se produisent sur deux lignes de vie différentes ($l_1$ et $l_3$).

La boucle permet qu'un nombre arbitraire d'instances de la réception de $m_3$ se produise de manière séquentielle. La co-région sur $l_2$ permet ensuite que la réception de $m_1$ ait lieu avant, entre ou après n'importe laquelle de ces instances de réception de $m_3$.

## 6.2 Solutions existantes

### 6.2.1 Approches pour apprendre des modèles formels.

Les algorithmes de process discovery (découverte de processus) [14] [15] analysent un journal d'événements (event log) pour en extraire un modèle formel — généralement un réseau de Petri [16] — qui représente le déroulement du processus observé et cherche à reproduire au mieux les séquences d'actions. L'évaluation de la similarité entre un modèle extrait et ses logs correspondant relève de l'analyse de conformance checking (vérification de conformité), qui inclut des critères tels que la proportion de traces reproduites par le modèle (*fitness*) et, inversement, la proportion de traces acceptées par le modèle mais absentes de l'event log (*précision*).

La découverte de processus stochastiques [17] étend les algorithmes de découverte "classiques" en utilisant la fréquence des traces observées pour extraire des modèles stochastiques à partir d'un journal d'événements.

Les objectifs des algorithmes de découverte de processus sont similaires à notre objectif d'inférence, mais ils traitent des réseaux de Petri de type workflow, et non des interactions.

L'article [18] propose une méthode d'apprentissage automatique pour générer des automates temporisés Uppaal à partir de traces. Bien que l'objectif soit similaire au nôtre, l'approche diffère, car elle ne prend pas en compte l'aspect textuel des traces pour prédire l'interaction (ou l'automate) en question.

Des travaux plus récents ont utilisé les réseaux de neurones pour générer des modèles formels à partir d'exemples, comme dans le cas de la synthèse d'expressions régulières [19].

### 6.2.2 Transformers pour les tâches de type séquence-à-séquence

L'architecture transformer [11–13] est une architecture d'apprentissage profond très populaire pour traiter les tâches Seq2Seq, établissant des résultats à l'état de l'art en traduction multilingue et en génération de texte. Elle est particulièrement pertinente pour les données séquentielles. Les transformers sont capables de comprendre et de générer des données ayant une structure séquentielle riche.

L'utilisation des transformers pour l'analyse de journaux d'événements a également démontré son efficacité, notamment pour la détection d'anomalies [20]. Par conséquent, l'utilisation des transformers pour exploiter des traces, qui partagent des caractéristiques similaires à celles des journaux d'événements, semble être une approche particulièrement adaptée.

### 6.2.3 HIBOU: Une librairie pour les interactions

Nous avons exploité l'outil HIBOU (Holistic Interaction Behavioral Oracle Utility) [21] développé par Erwan MAHE. Cet outil rassemble des utilitaires pour concevoir, dessiner, manipuler des modèles d'interaction, explorer leur sémantique et analyser les sorties de systèmes distribués (ensembles de journaux distribués) par rapport aux spécifications formelles écrites sous forme de modèles d'interaction.

En particulier voici les scripts que nous avons utilisé:

- `rng_gen_interactions` pour générer des interactions aléatoires (cf. génération de données 7.1)
- `analyze` pour tester si une trace est acceptée par une interaction (cf. évaluation 7.7)
- `explore` pour éxecuter des interactions, générer des traces, et créer des diagramme de séquence.
- `get_metrics` pour estimer certaines métriques des interactions de référence (cf. 7.6)
- `inclusion_checker` pour tester l'inclusion d'une interaction dans une autre (cf. évaluation 7.7)

# 7 Points clés et choix techniques

## 7.1 Génération de données synthétiques

Pour entraîner un modèle d'apprentissage profond, il est crucial de sélectionner judicieusement les données d'entrainement. En apprentissage profond, soit on utilise des bases de données, généralement annotées par des experts humains [22], ou bien on écrit une procédure pour générer des données par ordinateur [23]. Les données sont alors dites synthétiques.

Dans notre cas, il n'existe pas de vaste base de données de modèles d'interactions, il est donc inévitable d'utiliser des données synthétiques. Nous avons utilisé l'outil HIBOU 7.6 qui implémente une méthode simple consistant à choisir aléatoirement un opérateur ou une action, et compléter l'interaction récursivement suivant cette logique.

Avec cette méthode, nous avons plusieurs hyperparamètres qui contrôlent l'apparence des exemples générés.

- La profondeur maximale du terme d'interaction
- Le nombre de lifelines
- Le nombre de messages
- La distribution de probabilité lors du tirage des opérateurs

Plus de diversité et une plus grande taille dans les exemples générées rend la tâche plus difficile. Pour calibrer ces paramètres, nous nous sommes référés aux 20 exemples de références, pour reproduire la distribution des exemples de reférence, dans l'objectif de pouvoir bien généraliser. La distribution de probabilité lors du tirage des opérateurs est rapportée en annexe 4.

Par ailleurs, d'autres paramètres peuvent influencer la génération des données. Parmi ceux-ci, on peut citer :

- **La stratégie de génération** : HIBOU permet d'utiliser différentes stratégies pour générer des traces, notamment `HCS` (Heuristic Coverage Search) et `DFS` (Depth-First Search). Ces stratégies déterminent l'ordre et la manière dont les interactions sont parcourues pour construire les traces, ce qui peut avoir un impact significatif sur la diversité et la structure des traces générées.
- **La canonisation des traces** : HIBOU offre également la possibilité de canoniser les traces, c'est-à-dire de les réécrire dans une forme normalisée afin de réduire les variations non significatives. Cette étape peut être utile pour améliorer la cohérence des données d'entraînement et limiter le bruit introduit par des permutations équivalentes d'interactions.

- **La répétition et la découpe des traces** : Il y aussi la possibilité de répéter les traces ou de les découper en plusieurs morceaux.

Toutes ces options correspondent à autant d'hyperparamètres qui seront explorés par la suite. Ils constituent des leviers pour affiner la génération de données et mieux adapter les jeux de données aux objectifs du modèle.

## 7.2 Choix du type de modèle

L'utilisation d'un modèle de traduction encodeur-décodeur pour prédire une interaction à partir d'un ensemble de traces est justifiée par plusieurs facteurs.

- Les traces commes les interactions sont représentées par du texte donc une approche NLP à base de transformer parait adaptée
- Les traces sont des séquences d'événements chronologiques interconnectés, et le modèle encodeur-décodeur, notamment basé sur des Transformers, est bien adapté pour capturer les dépendances à long terme et les relations contextuelles dans ces données.
- le modèle peut générer des sorties structurées tout en apprenant de manière end-to-end, cela signifie qu'il est possible de former un modèle à partir des données brutes et d'obtenir directement des prédictions utiles sans avoir à concevoir explicitement de règles.

## 7.3 Encodage des traces et interactions

### 7.3.1 Encodage des traces

Une trace est originellement de la forme suivante, où les lignes de vie ($l1, l2$) envoient (!) ou reçoivent (?) des messages ($m1, m2, m3$) :

```
{ [l1, l2]  l2!m3.l1?m3.l2!m3.l1?m3.l2!m3.l1?m3.l2!m3.l1?m3.l2!m3.l1?m3.l1!m3.l1?m3.l1!m1.l1?m1}
```

Dans cette trace par exemple :

- $l2!m3$ indique que la ligne de vie $l2$ envoie le message $m3$.
- $l1?m3$ indique que la ligne de vie $l1$ reçoit le message $m3$.

Nous avons choisi de modifier très légèrement les traces, car celles-ci sont déjà très compactes et semblent adaptées à l'utilisation d'un modèle Transformer. Voici les deux options d'encodage envisagées :

- **Option 1 :** Supprimer les identifiants des lifelines et messages (l et m) et former des termes sous la forme $i!j$ ou $i?j$, où $i$ représente l'émetteur et $j$ le récepteur. Dans ce cas, la trace devient :

  ```
  2!3.1?3.2!3.1?3.2!3.1?3.2!3.1?3.2!3.1?3.1!3.1?3.1!1.1?1
  ```

  Ici, les identifiants des lifelines $l1, l2$ ont été remplacés par $1, 2$ et les messages $m1, m2, m3$ par $1, 2, 3$. Chaque terme $i!j$ ou $i?j$ fait partie du vocabulaire qui sera appris par le modèle.

- **Option 2 :** Garder l'encodage original des lifelines et des messages, et surtout les considérer comme des mots. Par exemple, on pourrait remplacer $l1, l2, l3$ par $a, b, c$ et $m1, m2, m3$ par $1, 2, 3$. La trace deviendrait alors :

  ```
  b.!.3.a.?.3.b.!.3.a.?.3.b.!.3.a.?.3.b.!.3.a.?.3.b.!.3.a.?.3.a.!.3.a.?.3.a.!.1.a.?.1
  ```

  Dans ce cas, $l1$ devient $a$, $l2$ devient $b$, et les messages $m1, m2, m3$ sont représentés par $1, 2, 3$. Ici, $a$, $b$, $c$, 1, 2, 3, ainsi que les symboles ! et ? font tous partie du vocabulaire qui sera appris par le modèle.

**Comparaison des deux options :**

- **Option 1 :** Cette méthode est plus simple et plus compacte, ce qui est utile pour réduire le nombre de termes dans une même trace. Cependant, elle perd l'information sur la structure des lifelines et des messages, ce qui peut compliquer l'apprentissage. De plus, cette méthode implique un vocabulaire de taille *nombre de lifelines* $\times$ *nombre de messages*, ce qui est moins avantageux que l'autre option.
- **Option 2 :** Cette méthode conserve la distinction entre les différentes lifelines et messages grâce à des mots raccourcis. En revanche, elle est plus verbeuse : un terme dans l'option 1 correspond à trois termes dans cette option, rendant les traces encodées trois fois plus longues. Elle implique un vocabulaire de taille *nombre de lifelines* + *nombre de messages*, ce qui permet de réduire significativement la taille du vocabulaire pour des interactions complexes avec un grand nombre de lifelines et de messages.

Les deux options sont intéressantes et nous les avons comparés dans nos expériences 8.2.

### 7.3.2 Encodage des interactions

**Option récursive**

Une interaction de base est notée comme suit : les opérateurs sont suivis de parenthèses et peuvent concerner plusieurs termes séparés par des virgules. Par exemple :

$$\text{seq}(m2 \to l1,\ m1 \to l2,\ l2 - -m3 \to |,\ l2 - -m3 \to |,\ \text{loopS}(\text{seq}(m1 \to l1,\ m2 \to l2)))$$

Nous avons décidé d'utiliser une **Reverse Polish Notation** (RPN) pour nous débarrasser des parenthèses, ce qui permet d'obtenir un texte plus compact sans perte d'information. Voici les deux options envisagées :

Les règles d'encodage sont appliquées de manière récursive avec les transformations suivantes :

| **Opérateur** | **Encodage** |
|---|---|
| $\text{strict}(a, b)$ | `a.b.st` |
| $\text{seq}(a, b)$ | `a.b.se` |
| $\text{par}(a, b)$ | `a.b.pr` |
| $\text{alt}(a, b)$ | `a.b.at` |
| $\text{loopS}(a)$ | `a.is` |
| $\text{loopW}(a)$ | `a.iw` |
| $\text{loopP}(a)$ | `a.ip` |

Pour les messages :

| **Message** | **Encodage** |
|---|---|
| $m2 \to l1$ | `1?2` |
| $l2 - -m1 \to \|$ | `2!1` |
| $l2 - -m1 \to l3$ | `2!1.3?1.st` |

Pour plusieurs opérandes, l'encodage est récursif :

$$\text{strict}(a, b, c) \to \texttt{a.b.c.st.st}, \quad \text{seq}(a, b, c) \to \texttt{a.b.c.se.se}, \quad \text{etc.}$$

**Exemple d'encodage pour l'exemple ci-dessus :**

`1?2.2?1.2!3.2!3.1?1.2?2.se.is.se.se.se.se`

**Option basée sur le nombre d'opérandes**

Dans cette option, le nombre d'opérandes est indiqué directement dans l'encodage. Les règles sont :

| Opérateur | Encodage |
|---|---|
| strict$(a, b)$ | `a.b.2st` |
| seq$(a, b)$ | `a.b.2se` |
| par$(a, b)$ | `a.b.2pr` |
| alt$(a, b)$ | `a.b.2at` |
| loopS$(a)$ | `a.1is` |
| loopW$(a)$ | `a.1iw` |
| loopP$(a)$ | `a.1ip` |

Pour les messages :

| Message | Encodage |
|---|---|
| $m2 \to l1$ | `1?2` |
| $l2 - -m1 \to \vert$ | `2!1` |
| $l2 - -m1 \to l3$ | `2!1.3?1.2st` |

Pour plusieurs opérandes, l'encodage est direct :

$$\text{strict}(a, b, c) \to a.b.c.3\text{st}, \quad \text{seq}(a, b, c) \to a.b.c.3\text{se}, \quad \text{etc.}$$

**Exemple d'encodage pour l'exemple ci-dessus :**

```
1?2.2?1.2!3.2!3.1?1.2?2.2se.1is.5se
```

**Comparaison des deux options**

- **Première option :** Plus détaillée et basée sur la récursivité, mais nécessite des cas spéciaux pour gérer plusieurs termes. Elle permet d'avoir un vocabulaire moins grand pour encoder les interactions, puisque chaque opérateur correspond à un seul mot. Cependant, elle implique un encodage plus long avec des répétitions.
- **Deuxième option :** Plus compacte pour des interactions complexes, elle permet d'éviter les répétitions. Toutefois, elle implique un vocabulaire plus grand pour encoder les interactions, dans la mesure où certains opérateurs, comme `seq`, nécessitent plusieurs mots possibles (`2se`, `3se`, `4se`, etc.).

Les deux options ont été implémentées et nous les avons comparés dans nos expériences 8.2.

## 7.4 Fonctionnement du modèle de traduction

### 7.4.1 Tokenisation, embedding et positionnal encoding

Avant de pouvoir entraîner un modèle, il faut tokeniser nos entrées et sorties. Pour cela, on prend chaque terme (séparé par des points) et on leur associe un token (un entier identifiant le terme) :

- Pour les traces en entrée : `"1!1<c>1?3<c>1?1.2?3<c>1!2<c>1!2.1?3<c>1!2.1?3.1?3"` → `[2, 6, 5, 11, 5, 7, 17, 5, 8, 5, 8, 11, 5, 8, 11, 11, 3]`
- Pour les interactions en sortie : `"2?3.ip.2?2.1?1.2?1.2!1.se.ip.se.se.se"` → `[ 2, 25, 11, 23, 15, 21, 20, 9, 11, 9, 9, 9, 3]`

Ensuite on va transformer nos tokens en **embedding** ce qui permet de convertir les tokens en vecteurs de nombres réels. Ces vecteurs d'embedding capturent les relations sémantiques entre les mots en fonction de leur contexte, permettant ainsi aux modèles d'apprentissage automatique de travailler sur des représentations continues des mots plutôt que sur des représentations discrètes.

Cependant, les modèles de type transformer n'ont pas de notion implicite d'ordre des mots dans une phrase. Pour résoudre ce problème, le **positionnal encoding** est utilisé. Le positional encoding consiste à ajouter un vecteur représentant la position de chaque mot dans la séquence à son vecteur d'embedding. Cela permet au modèle de différencier les mots en fonction de leur position dans la phrase et d'apprendre des relations contextuelles tout en tenant compte de l'ordre des tokens.

### 7.4.2 Architecture du modèle

Le modèle 3 se compose de deux composants principaux : un **encodeur** et un **décodeur**.

L'**encodeur** reçoit les traces en entrée et en apprend la signification. Il génère ensuite un vecteur qui représente ces traces.

Le **décodeur**, quant à lui, prend ces traces vectorisées ainsi qu'un début d'interaction et utilise cette information pour prédire le prochain mot de l'interaction.

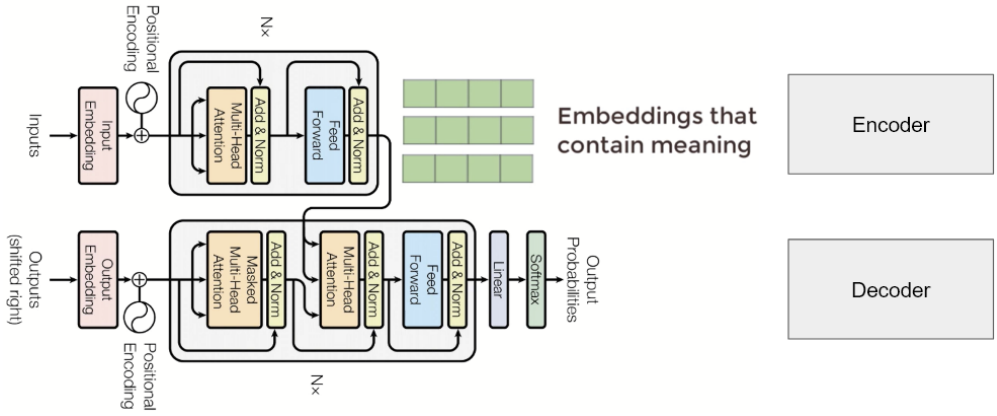



Figure 3: Architecture du modèle [12]

### 7.4.3 Entrainement du modèle

Nous disposons d'un ensemble de traces associées à leurs interactions correspondantes. Le modèle prédit les mots de l'interaction, puis ces prédictions sont comparées aux mots réels pour entraîner le modèle, afin qu'il puisse générer les bons mots.

- **Entrées de l'encodeur** : L'encodeur prend comme entrée un ensemble de tokens qui représente des traces.
- **Sortie de l'encodeur** : L'encodeur renvoie l'embedding de ces traces.
- **Entrées du décodeur** : Le décodeur prend comme entrée l'embedding des traces ainsi que quelques mots déjà générés de l'interaction (ou des mots réels dans le cadre de l'entraînement).
- **Sortie du décodeur** : Le décodeur prédit un mot (ou token) à chaque étape, en s'appuyant sur l'embedding des traces et sur les mots précédemment générés.

L'entraînement se fait par prédiction des mots de l'interaction, étape par étape. Pour chaque mot généré, le modèle compare sa prédiction avec le mot réel de l'interaction. Cette comparaison est effectuée à l'aide d'une fonction de coût, la *cross-entropy*, qui mesure la différence entre la probabilité prédite pour le mot, et le mot cible.

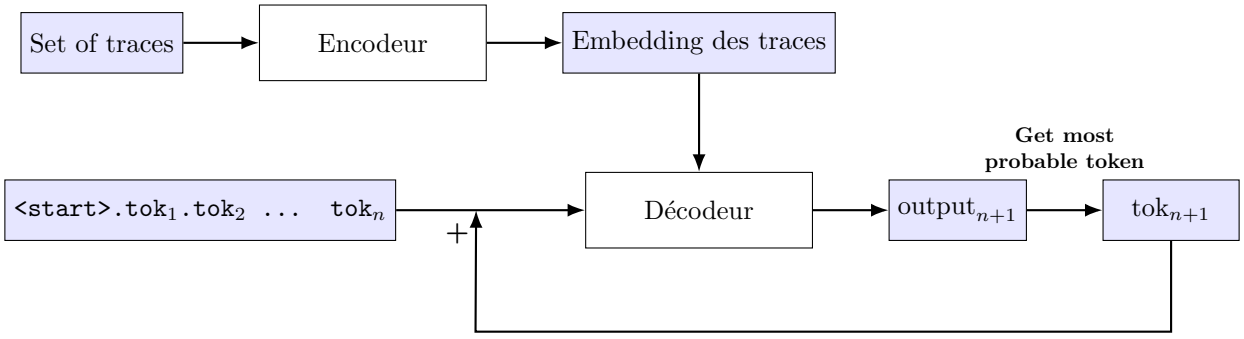
Un algorithme de rétropropagation est utilisé pour ajuster les paramètres du modèle, en minimisant l'erreur entre les mots prédits et les mots réels. L'objectif est d'entraîner le modèle à prédire avec précision les mots suivants dans l'interaction.

## 7.5 Hyperparamètres du modèle

Les hyperparamètres du modèle Transformer utilisés dans ce projet sont les suivants :

- `embedding_size` : correspond à la dimension des vecteurs de représentation (ou embeddings) utilisés pour encoder les éléments en entrée. Plus cette dimension est grande, plus le modèle peut capturer des informations complexes, au prix d'un coût computationnel plus élevé.
- `nhead` : représente le nombre de têtes d'attention dans le mécanisme d'attention multi-tête. Chaque tête apprend à se concentrer sur différentes parties de la séquence, permettant au modèle de capter plusieurs relations en parallèle.
- `dim_feedforward` : taille de la couche feedforward située après le mécanisme d'attention dans chaque bloc Transformer. Elle contrôle la capacité d'apprentissage non linéaire du modèle.
- `num_encoder_layers` : nombre de couches dans l'encodeur du Transformer. Chaque couche contient un mécanisme d'attention et un réseau feedforward, empilés pour permettre au modèle de capturer des relations hiérarchiques dans les données.
- `num_decoder_layers` : nombre de couches dans le décodeur du Transformer, fonctionnant de manière similaire à l'encodeur, mais en intégrant également l'attention sur les sorties de l'encodeur.

### 7.5.1 Inférence du modèle




Pour l'inférence du modèle, on donne en entrée à notre encodeur l'ensemble de tokens représentant un ensemble de traces. Cela permet d'obtenir un *embedding* de ces traces. Ensuite, on fournit au décodeur l'*embedding* obtenu ainsi que le token `<start>` qui indique le début d'une interaction.

Le modèle prédit alors un vecteur contenant une probabilité pour chaque token possible. On choisit ensuite le token le plus probable, noté $\text{tok}_1$. On recommence ensuite en donnant au décodeur l'*embedding* des traces et la séquence `<start>`.$\text{tok}_1$. Le modèle prédit alors le token suivant $\text{tok}_2$, et ainsi de suite.

Ce processus se répète jusqu'à ce que le modèle prédise le token `<end>` (indiquant la fin de l'interaction), ou jusqu'à ce qu'un nombre maximal de tokens, fixé au préalable, soit atteint.

### 7.5.2 Limitation du vocabulaire

Un problème que nous avons rencontré est que le modèle prédisait parfois des interactions avec des lignes de vie ou des messages qui n'étaient pas présents dans les traces. Pour remédier à ce problème, nous avons utilisé un masque pour limiter le vocabulaire.

À partir de l'ensemble de traces en entrée, un masque est défini pour indiquer quels tokens sont possibles à prédire pour l'interaction. Le modèle prédit alors un vecteur contenant une probabilité pour chaque token possible. Le masque est appliqué et les tokens impossibles voient leur probabilité devenir nulle. Nous choisissons ensuite le token ayant la probabilité la plus élevée, ce qui permet de garantir que l'interaction générée ne contiendra pas d'éléments non présents dans les traces.

## 7.6 Collecte d'exemples de référence

Pour évaluer notre approche dans des cas d'usage réels, nous avons cherché des bases de données d'interactions utilisées en guise de benchmarks. Malheuresement, ce genre de base de données n'existe pas, bien que cela existe pour des cas similaires comme des regex avec le Polyglot Regex Corpus [24].

Nous avons donc collecté 20 interactions de référence retrouvées dans la littérature et suggérées par nos encadrants. Les exemples et leurs sources sont détaillés en annexe 5.

Afin de générer des données synthétiques ressemblantes à des données réelles, nous avons procédé à une analyse de données rudimentaire pour étudier quelle est la taille typique d'une interaction.

| **Métrique** | **Moyenne** | **Médiane** |
|---|---|---|
| Profondeur | 3.9 | 4 |
| Nombre de lifelines | 3.8 | 3 |
| Nombre de messages | 7.1 | 5 |
| Nombre d'actions | 21.5 | 16 |

Table 1: Valeurs moyennes et médianes de certaines métriques dans les 20 exemples réels.

| **Opérateur** | **Fréquence** |
|---|---|
| seq | 0.408 |
| par | 0.203 |
| alt | 0.188 |
| loopS | 0.139 |
| strict | 0.052 |
| loopP | 0.009 |

Table 2: Fréquence des opérateurs dans les 20 exemples réels.

## 7.7 Métriques d'évaluation du modèle

Durant la phase d'entraînement, le transformer reçoit des couples traces-interaction $(I, T)$, et il est entraîné à prédire une interaction $I_{pred} = f_\theta(T)$, en connaissant seulement les traces $T = \{t_1, \ldots, t_p\}$. Les paramètres $\theta$ du réseau sont actualisés en minimisant la *cross-entropy loss* entre sa prédiction $f_\theta(T)$ et l'interaction $I$, et par descente de gradient.

Cependant, cette *cross-entropy loss* n'indique pas parfaitement la performance du transformer à l'inférence et n'est pas interprétable. Plusieurs propositions d'interactions sont équivalentes, c'est-à-dire qu'elles acceptent toutes les traces acceptées par $I$. Si le transformer prédit une alternative équivalente valide à l'interaction, il est quand même pénalisé par la *cross-entropy loss*.

Nous avons donc souhaité introduire de nouvelles métriques qui manquait pour mieux formaliser notre problème et notre objectif de prédiction, et qui offre une meilleure interprétabilité sur la performance du modèle.

**Definition 7.1: Traces acceptées par une interaction**

Un trace $t$ est acceptée par une interaction $I$ si elle peut être obtenue par éxectution de l'interaction $I$. On note $T(I)$ l'ensemble des traces acceptées par l'interaction $I$

### 7.7.1 Fitness et Précision

Nous nous sommes inspirés de [17], pour définir la *fitness* et la *précision*:

- La *fitness* quantifie la capacité à prédire une interaction qui accepte bien les traces données.
- La *précision* contrôle le fait que l'interaction prédite ne soit pas trop acceptante comparée à la vraie interaction.

**Definition 7.2: Fitness et Précision**

Soit $I$ une interaction, $T \subseteq T(I)$ les traces d'entrée et $I_{pred} = f_\theta(T)$ la prédiction du transformer. La *fitness* $\text{Fit}(I_{pred}, I)$ est la proportion des traces acceptées par $I$ qui sont acceptées par $I_{pred}$.

$$\text{Fit}(I_{pred}, I) = \frac{1}{\#T(I)} \#\{t \in T(I) \text{ et } t \text{ est accepté par } I_{pred}\}$$

La *précision* est la proportion des traces acceptées par $I_{pred}$ qui sont acceptées par $I$.

$$\text{Pr}(I_{pred}, I) = \frac{1}{\#T(I_{pred})} \#\{t \in T(I_{pred}) \text{ et } t \text{ est accepté par } I\}$$

Pour évaluer un modèle $f_\theta$ sur un ensemble de test $\mathcal{D}_{\text{test}}$, on calcule la moyenne de la fitness (resp. la précision) sur toutes les instances de test $(I_k, T_k)$.

$$\text{Fit}(\theta) = \frac{1}{\#\mathcal{D}_{\text{test}}} \sum_{(I_k, T_k) \in \mathcal{D}_{\text{test}}} \text{Fit}(f_\theta(T_k), I_k)$$

$$\text{Pr}(\theta) = \frac{1}{\#\mathcal{D}_{\text{test}}} \sum_{(I_k, T_k) \in \mathcal{D}_{\text{test}}} \text{Pr}(f_\theta(T_k), I_k)$$

En particulier, si la prédiction $I_{pred}$ est équivalente à $I$, alors $\text{Fit}(I_{pred}, I) = \text{Pr}(I_{pred}, I) = 1$. Si la prédiction n'accepte aucune trace d'entrée, $\text{Fit}(I_{pred}, I) = 0$. Si la prédiction accepte toutes les traces possibles, alors $\text{Pr}(I_{pred}, I) = 0$

Au cours de notre étude, on cherche les paramètres $\theta$ qui maximisent la combinaison de la fitness et de la précision. On introduit le Mean-Score (mean score) et le F-Score (f1-score):

**Definition 7.3: Mean-Score et F-Score**

$$\text{Mean-Score}(\theta) = \frac{\text{Fit}(\theta) + \text{Pr}(\theta)}{2}, \text{F-Score}(\theta) = 2 \times \frac{\text{Fit}(\theta) \times \text{Pr}(\theta)}{\text{Fit}(\theta) + \text{Pr}(\theta)}$$

Le M-Score constitue notre objectif principal pendant ce projet. Nous avons constaté dans de très nombreux cas que la *fitness* et la *précision* ont des valeurs proches.

### 7.7.2 Autres métriques

Nous avons regardé plusieurs autres métriques suggérées par nos encadrants.

Nous avons évalué les interactions prédites par la tree-edit distance [25], une généralisation de l'edit distance aux arbres:

La tree-edit distance entre deux arbres ordonnés et étiquetés A et B, notée $d_{tree}(A, B)$, correspond au nombre minimal d'opérations pour transformer l'arbre $A$ en l'arbre $B$. Il y a 3 opérations d'édition sur des arbres ordonnés et étiquetés:

- renommer l'étiquette d'un noeud.
- supprimer un nœud et connecter ses enfants à son parent en maintenant l'ordre ;
- insérer un noeud entre un noeud existant et une sous-séquence d'enfants consécutifs de ce nœud ;

On utilise une représentation de l'interaction $I$ (resp. $I_{pred}$) en un arbre $A$ (resp. $A_{pred}$) pour calculer la tree-edit distance entre les arbres $A$ et $A_{pred}$ et on normalise pour garantir que le résultat est entre 0 et 1.

**Definition 7.4: Edit-Score**

L'Edit-score est la tree edit distance entre la prédiction et la target, normalisée entre 0 et 1:

$$\text{Edit-Score}(I_{pred}, I) = \frac{d_{tree}(A_{pred}, A)}{\max(N(A_{pred}), N(A))},$$

avec $N(A)$ le nombre de nœuds de l'arbre $A$. De façon analogue à la *fitness* et la *précision*, l'Edit-score d'un modèle est calculé par une moyenne sur l'ensemble de test.

$$\text{Edit-Score}(\theta) = \frac{1}{\#\mathcal{D}_{\text{test}}} \sum_{(I_k, T_k) \in \mathcal{D}_{\text{test}}} \text{Edit-Score}(f_\theta(T_k), I_k)$$

Nous avons également évalué nos modèles sur la proportion des interactions prédites qui sont incluses dans l'interaction $I$, et respectivement la proportion des interactions prédites qui incluent l'interaction $I$. On note ces métriques "% Includes" et "% Included". On rappelle qu'une interaction $I$ inclut une interaction $I'$ si $T(I') \subseteq T(I)$.

# 8 Expériences

## 8.1 Protocole

Pour chaque expérimentation, nous générons trois ensembles de données distincts : un ensemble d'entraînement (`train`), un ensemble de validation (`valid`) et un ensemble de test (`test`). Ces ensembles sont construits à partir du jeu de données initial, selon une répartition de taille : `size`, `size // 5` et `size // 5`, respectivement.

Le modèle est entraîné sur l'ensemble `train`, avec une évaluation régulière de ses performances sur l'ensemble `valid`. Cette validation intermédiaire permet de suivre la progression de l'apprentissage et d'ajuster, le cas échéant, les hyperparamètres ou les stratégies d'apprentissage. Une évaluation finale est effectuée sur `test` à la fin de l'entraînement, afin d'estimer la capacité de généralisation du modèle sur des données totalement inédites.

Nous utilisons également une stratégie d'*early stopping* : si les performances du modèle sur l'ensemble de validation n'évoluent plus significativement pendant 5 époques consécutives, l'entraînement est arrêté prématurément. Cela permet d'éviter le surapprentissage et de gagner du temps de calcul lorsque le modèle converge rapidement.

Nous avons exploré deux configurations principales pour les données , définies par les paramètres `max_lifelines`, `max_messages` et `max_depth`, toutes deux suivant une distribution d'opérateurs observée à partir des exemples de références recueillis 7.6:

- **Configuration 1** : `max_lifelines = 3`, `max_messages = 7`, `max_depth = 3`, `distrib = typical_case`
- **Configuration 2** : `max_lifelines = 4`, `max_messages = 7`, `max_depth = 4`, `distrib = typical_case`

La première configuration est volontairement peu ambitieuse. Elle a été pensée comme un point de départ simple, avec l'idée que, combinée à un modèle capable de regrouper plusieurs petites interactions en une plus complexe, elle pourrait suffire à prédire des interactions de plus grande taille. Cette piste a été proposée par nos encadrants, dans une optique de généralisation.

La seconde configuration représente un cas plus réaliste, correspondant à un niveau de complexité moyen observé dans les exemples réels que nous avons recueillis. Elle vise à refléter des interactions plus représentatives du domaine d'application, et une plus grande diversité.

## 8.2 Exploration des hyperparamètres

Afin d'optimiser les performances de notre modèle, nous avons exploré un certain nombre d'hyperparamètres à travers différentes expérimentations. Cette phase a été essentielle pour identifier les combinaisons les plus

pertinentes dans notre contexte. Le tableau ci-dessous résume les différents hyperparamètres testés, les valeurs explorées pour chacun d'eux, ainsi que la ou les valeurs retenues comme étant les plus efficaces :

| Catégorie | Hyperparamètre | Valeurs explorées | Meilleure valeur |
|---|---|---|---|
| **Entraînement** | `batch size` | (16, 32) | 16 |
| | `learning rate` | (5e-4, 1e-4, 5e-5, 1e-5, 1e-6) | 1e-4 |
| | `epochs` | (20, 30, 40, 50, 100) | 40 |
| **Données** | `interaction encoding` | (basic, op_num) | basic |
| | `trace encoding` | (basic, detailed) | basic |
| | `message occurrence type` | (inv, num_order, 0-rep) | num_order |
| | `trace generation strategy` | (HCS, DFS) | HCS |
| | `canonize traces` | (true, false) | true |
| | `size of dataset` | (10k, 50k, 200k) | 50k ou 200k |
| | `prefix size` | (-1, 10, 20) | -1 |
| | `min number of traces` | (0, 10, 20, 50) | 20 ou 50 |
| | `max number of traces` | (10, 20, 50) | 50 |
| **Architecture modèle** | `embedding size` | (100, 256, 512, 1024) | 256 |
| | `nhead` | (4, 8) | 8 |
| | `dim feedforward` | (100, 256, 512, 1024, 2048, 4096) | 1024 |
| | `num encoder layers` | (2, 3, 4) | 2 |
| | `num decoder layers` | (2, 3, 4) | 2 |
| | `mask inference` | (true, false) | false |

Ces hyperparamètres ont été sélectionnés à l'aide d'une recherche par grille (*grid search*) avec Ray Tune sur la configuration 2. Les meilleures combinaisons ont été déterminées en fonction des résultats sur les ensembles de validation. La signification de chacun des hyperparamètres est détaillé section 7.

## 8.3 Performance

| Configuration | Valid Fitness | Valid Mean Score | Valid Precision | Train Loss | Valid Loss | Test F1 Score | Test Fitness | Test Loss | Test Mean Score | Test Precision |
|---|---|---|---|---|---|---|---|---|---|---|
| Config 1 | 0.91 | 0.92 | 0.95 | 0.16 | 0.11 | 0.86 | 0.87 | 0.16 | 0.89 | 0.91 |
| Config 2 | 0.57 | 0.58 | 0.61 | 0.68 | 0.71 | 0.54 | 0.54 | 0.53 | 0.54 | 0.55 |

Table 3: Meilleure performance sur les ensembles de test et de validation

Les résultats obtenus pour la configuration 1, la plus simple, sont très satisfaisants avec un F1-score de 0,86, indiquant que la majorité des interactions sont correctement prédites. En revanche, la configuration 2 présente des performances plus modestes, avec un F1-score de 0,54, ce qui signifie que de nombreuses interactions sont mal ou partiellement prédites. Dans un cas d'usage concret, cette configuration pourrait fournir à un expert une validation partielle de ses hypothèses, mais ne serait pas suffisamment fiable pour être utilisée de manière autonome.

## 8.4 Corrélation entre les métriques et la loss d'entraînement

Nous avons souhaité vérifier que l'apprentissage du transformer donne systématiquement lieu à l'amélioration des métriques, et qu'il n'y a pas de "mismatch", au sens où une loss d'entraînement plus basse donnerait de moins bonnes métriques.

Nous avons démontré qu'il y a une forte corrélation entre la closs d'entraînement et les métriques de validation. Pour ce faire, nous avons réalisé 10 entraînements avec différents hyperparamètres du réseau, avec le même ensemble de test, et nous aovns a évalué le réseau à l'époque 10, 20 et 30. On observe, sur les figures 4 que lorsque la loss d'entraînement baisse, toutes les métriques sur l'ensemble de validation tendent à s'améliorer.

# 9 Conclusion

Nous avons mis en place une pipeline complète et modulable de génération de données à l'aide de HIBOU, puis entraîné des modèles Transformers capables de prédire efficacement les interactions à partir de traces. Nos résultats montrent que l'utilisation de Transformers pour cette tâche constitue une approche prometteuse. Cependant, nous avons également identifié certaines limites, notamment en lien avec la taille et la profondeur des interactions.

Étant donné que les exemples réels présentent souvent des interactions de grande taille, notre modèle n'est pas directement applicable à des cas concrets. Il peut néanmoins constituer un outil d'aide à la décision pour un expert.

Ce projet s'inscrit dans une démarche de recherche, et plusieurs pistes d'amélioration pourraient être explorées à partir de notre travail :

- Entraîner d'autres types de modèles en tirant parti de la flexibilité de notre pipeline de génération de données, afin d'explorer des alternatives aux transformers.
- Utiliser de plus grands datasets et une plus grande capacité de calcul.
- Combiner notre modèle avec un autre capable de regrouper de petites interactions en une interaction plus large, d'après une idée de Kazuma IKESAKA.
- Développer des modèles spécialisés, entraînés sur des sous-ensembles de données restreints (par exemple, des jeux de données contenant exactement $n$ *lifelines* et $m$ messages par interaction), puis utiliser ces modèles de façon ciblée en détectant automatiquement $n$ et $m$ dans les traces au moment de l'inférence.
- Proposer plus d'explicabilité en inspectant les attention maps du transformer.

# References


[1] E. Mahe, B. Bannour, C. Gaston, A. Lapitre, and P. Le Gall, "Finite automata synthesis from interactions," 2024.

[2] E. Mahe, B. Bannour, C. Gaston, A. Lapitre, and P. L. Gall, "Dealing with observability in interaction-based offline runtime verification of distributed systems," 2022.

[3] J. Lange, E. Tuosto, and N. Yoshida, "From communicating machines to graphical choreographies," 2015.

[4] M. Sun, F. Arbab, and C. Baier, "Synthesis of reo circuits from scenario-based interaction specifications," *Sci. Comput. Program.*, vol. 76, pp. 651–680, 08 2011.

[5] D. Ancona, A. Ferrando, L. Franceschini, and V. Mascardi, "Coping with bad agent interaction protocols when monitoring partially observable multiagent systems," in *PAAMS*, pp. 59–71, 2018.

[6] S. A. W. Matt Lyke, Betty C Nguyen, "dbay info 620- information systems analysis & design," 2013.

[7] T. Jensen, "Updated SPARTA SRIA (Roadmap v3)," tech. rep., INRIA, Sept. 2022.

[8] M. Bakillah, S. Liang, A. Zipf, and M. A. Mostafavi, "A dynamic and context-aware semantic mediation service for discovering and fusion of heterogeneous sensor data," *Journal of Spatial Information Science*, vol. 6, pp. 155–185, 06 2013.

[9] L. Edixhoven, S.-S. Jongmans, J. Proença, and I. Castellani, "Branching pomsets: design, expressiveness and applications to choreographies," *Journal of Logical and Algebraic Methods in Programming*, vol. 136, p. 100919, 2024.

[10] E. Elemam, A. Bahaa-Eldin, N. Hamdy, and M. Sobh, "Formal verification for a pmqtt protocol," *Egyptian Informatics Journal*, vol. 21, 02 2020.

[11] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[13] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao, "Learning deep transformer models for machine translation," July 2019.

[14] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs - a constructive approach," in *Application and Theory of Petri Nets and Concurrency* (J.-M. Colom and J. Desel, eds.), (Berlin, Heidelberg), pp. 311–329, Springer Berlin Heidelberg, 2013.

[15] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[16] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[17] P. Cry, A. Horváth, P. Ballarini, and P. Le Gall, "A framework for optimisation based stochastic process discovery," in *Quantitative Evaluation of Systems and Formal Modeling and Analysis of Timed Systems*, (Cham), pp. 34–51, Springer Nature Switzerland, 2024.

[18] J. Vain, G. Kanter, and A. Anier, "Learning timed automata from interaction traces**this research was partially supported by the estonian ministry of education and research institutional research grant no iut33-13.," *IFAC-PapersOnLine*, vol. 52, no. 19, pp. 205–210, 2019. 14th IFAC Symposium on Analysis, Design, and Evaluation of Human Machine Systems HMS 2019.

[19] S.-H. Kim, H. Cheon, Y.-S. Han, and S.-K. Ko, "Neuro-symbolic regex synthesis framework via neural example splitting," 2022.

[20] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, "Hitanomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2064–2076, 2020.

[21] E. Mahe, *An operational semantics of interactions for verifying partially observed executions of distributed systems*. Theses, Université Paris-Saclay, July 2021.

[22] Y. Jia, M. Tadmor Ramanovich, Q. Wang, and H. Zen, "CVSS corpus and massively multilingual speech-to-speech translation," in *Proceedings of Language Resources and Evaluation Conference (LREC)*, pp. 6691–6703, 2022.

[23] L. Long, R. Wang, R. Xiao, J. Zhao, X. Ding, G. Chen, and H. Wang, "On llms-driven synthetic data generation, curation, and evaluation: A survey," 2024.

[24] J. C. Davis, L. G. M. IV, C. A. Coghlan, F. Servant, and D. Lee, "Why aren't regular expressions a lingua franca? an empirical study on the re-use and portability of regular expressions," 2021.

[25] M. Pawlik and N. Augsten, "Tree edit distance." `http://tree-edit-distance.dbresearch.uni-salzburg.at/`, 2016. University of Salzburg. Accessed: 2025-04-10.

[26] E. Mahe, B. Bannour, C. Gaston, A. Lapitre, and P. L. Gall, "A term-based approach for generating finite automata from interaction diagrams," 2023.

# 10 Annexes

## 10.1 Distribution des données synthétique

| Hyperparamètre | Value |
|---|---|
| P(empty) | 0.025 |
| P(action) | 0.05 |
| P(transmission) | 0.225 |
| P(seq) | 0.3 |
| P(alt) | 0.15 |
| P(par) | 0.15 |
| P(loopS) | 0.1 |

Table 4: Distribution de probabilité utilisée à la génération.

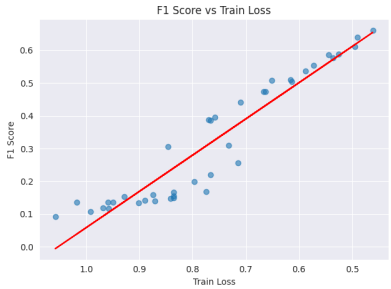
## 10.2 Sources des exemples de référence

| Nom de l'interaction | Description | Source | Symboles |
|---|---|---|---|
| Alternating Bit Protocol | A network protocol operating at the data link layer | [26] | 182 |
| ATM protocol | Verification, refusal, and acceptance of withdrawals | [9] | 113 |
| Buyer-Seller | Interaction between a seller and a buyer | - | 63 |
| Cloud | Interaction between subsystems of a cloud application | - | 27 |
| Human-Resources | A DApp system for managing Human Resources | [26] | 83 |
| MQTT | Publish-subscribe messaging based on the TCP/IP protocol | [10] | 84 |
| Online Banking | Identification in online banking | [4] | 108 |
| PayPal | Buyer verification on PayPal | [6] | 47 |
| Platoon | A connected platoon of autonomous rovers | [7] | 60 |
| Sensors | A network of communicating sensors components | [8] | 79 |
| Voting protocol | Asymmetric voting protocol with three actors | [9] | 82 |
| Book-Order | A protocol with 5 actors for purchasing books | [5] | 41 |
| Buyer-Seller (choreo) | Scenario with Buyer and Seller | [9] | 50 |
| Buyer-Seller (variant) | Buyer and Seller communicating via an ill channel | [9] | 52 |
| Seller-Seller (variant) | Two sellers communicating via an ill channel | [9] | 40 |
| Choregraphy 1 | Choregraphy included in the paper | [9] | 26 |
| Choregraphy 2 | Choregraphy included in the paper | [9] | 26 |
| Race | Two runners in a race with a controller. | [9] | 52 |
| Strict Review | C waits for A and B to reply before sending a confirmation | [9] | 54 |
| Streaming Protocol | Real-time streaming interaction between modules | [9] | 36 |

Table 5: Les 20 exemples d'interactions de référence utilisés.

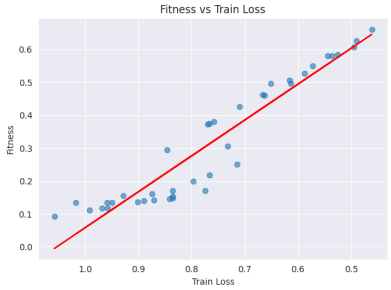
## 10.3 Corrélations entre les métriques d'évaluation et la loss d'entrainement


(a) Mean-Score vs Train Loss

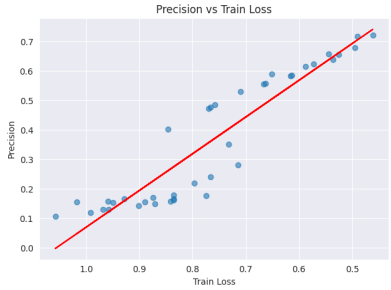



(b) Fitness vs Train Loss

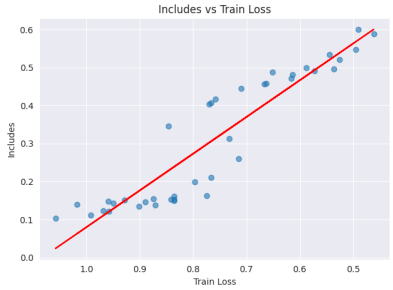

(c) Edit-Score vs Train Loss



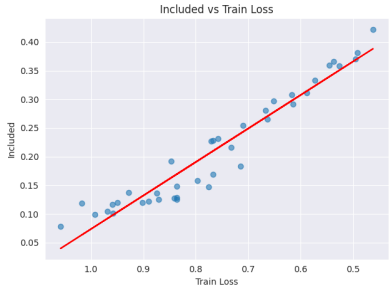

(d) Fitness vs Train Loss




(e) Precision vs Train Loss




(f) % Includes vs Train Loss




(g) % Included vs Train Loss

Figure 4: Corrélations entre les métriques sur l'ensemble de validation (axe Y) et la loss d'entrainement (axe X). Pour collecter ces données, nous avons réalisé 10 entrainements avec différents hyperparamètres du réseau, avec le même ensemble de validation, et on a évalué le réseau à l'époque 10, 20 et 30.