Mise à l'épreuve des Chatbots

Quelles sont les limites d'un chatbot dans la réalisation d'un projet informatique ?

Étude de cas : Jeu du Démineur en Python

Jordan Matin

Université Libre de Bruxelles

INFO-F-106 : Projet d'Informatique

14 mai 2023

Résumé

Ce travail analyse l'utilité de ChatGPT, un chatbot basé sur un Large Language Model (LLM), dans le cadre du développement d'un projet informatique : le jeu du Démineur en Python. Nous évaluons sa capacité à générer du code fonctionnel, à corriger des erreurs, à optimiser certaines parties du programme et à s'adapter aux contraintes imposées par l'utilisateur.

Nos résultats montrent que ChatGPT peut accélérer certaines étapes du développement, notamment la génération de fonctions spécifiques, la transformation de code itératif en code récursif et la correction de bugs. Cependant, son efficacité dépend fortement de la précision des instructions fournies ainsi que du niveau d'expertise du programmeur.

Une utilisation naïve, sans encadrement ni esprit critique, conduit à des résultats superficiels, incomplets ou incorrects. En revanche, une utilisation méthodique et critique peut constituer un véritable outil d'assistance pour le développeur expérimenté. Nous concluons que ChatGPT ne remplace pas le programmeur, mais peut servir d'assistant précieux lorsqu'il est utilisé intelligemment.

Table des matières

1 Introduction

1.1 Contexte et motivations

La popularité des chatbots basés sur des modèles de langage de grande taille (*Large Language Models*, LLM) a connu une croissance spectaculaire ces dernières années. Ces systèmes sont capables de produire du texte cohérent, de répondre à des questions complexes, de résumer des documents et même de générer du code informatique dans de nombreux langages de programmation.

Parmi ces outils, ChatGPT, développé par OpenAI, s'est imposé comme une référence. Lancé en novembre 2022, il a rapidement attiré l'attention du grand public et des professionnels, suscitant à la fois enthousiasme et interrogations quant à son potentiel et ses limites.

Dans le cadre du cours INFO-F-106, nous avons été amenés à développer un jeu du Démineur en Python. Ce projet constitue un excellent cas d'étude pour évaluer les capacités réelles de ChatGPT en matière de génération de code. L'objectif de ce rapport est donc d'analyser, de manière scientifique et critique, dans quelle mesure un chatbot comme ChatGPT peut assister — voire remplacer — un programmeur dans la réalisation d'un projet informatique.

1.2 Qu'est-ce qu'un Large Language Model (LLM) ?

Un **Large Language Model** (LLM) est un modèle d'intelligence artificielle entraîné sur d'énormes quantités de données textuelles, souvent de l'ordre de plusieurs centaines de milliards de mots. L'objectif d'un LLM est de prédire le mot suivant dans une séquence de texte, ce qui lui permet de générer des réponses cohérentes et contextuellement pertinentes.

Les LLMs modernes reposent sur une architecture appelée **Transformer**, introduite en 2017 par Vaswani et al. dans l'article fondateur *"Attention Is All You Need"* [1]. Cette architecture a révolutionné le domaine du traitement du langage naturel (*Natural Language Processing*, NLP) grâce à un mécanisme innovant appelé **self-attention**.

1.2.1 Le mécanisme d'attention

Le mécanisme d'attention permet au modèle de pondérer l'importance de chaque mot dans une phrase par rapport aux autres mots. Contrairement aux architectures précédentes (comme les réseaux de neurones récurrents, ou RNN), le Transformer traite tous les mots d'une phrase simultanément, ce qui améliore considérablement les performances et la rapidité d'entraînement.

De manière simplifiée, pour chaque mot d'une phrase, le modèle calcule un score d'attention avec tous les autres mots, ce qui lui permet de capturer des relations sémantiques complexes, même entre des mots éloignés dans la phrase.

1.2.2 La tokenisation

Avant d'être traité par un LLM, le texte doit être converti en une séquence de **tokens**. Un token peut représenter un mot, une partie de mot ou un caractère de ponctuation. Chaque token est ensuite associé à un identifiant numérique, permettant au modèle de manipuler le texte sous forme de vecteurs mathématiques.

Par exemple, la phrase *"Bonjour le monde !"* pourrait être tokenisée en :

`['Bonjour', 'le', 'monde', '!']` → `[1523, 67, 892, 2]`

1.3 Qu'est-ce que ChatGPT ?

ChatGPT est une application conversationnelle développée par OpenAI, basée sur la famille des modèles GPT (*Generative Pre-trained Transformer*). Ces modèles sont :

1. **Pré-entraînés** sur de vastes corpus textuels issus d'Internet (livres, articles, forums, code source, etc.).
2. **Ajustés** (*fine-tuned*) pour suivre des instructions humaines grâce à une technique appelée *Reinforcement Learning from Human Feedback* (RLHF) [5].

La version utilisée pour ce travail est basée sur GPT-3.5, qui compte environ 175 milliards de paramètres. À titre de comparaison, GPT-2 (2019) comptait 1,5 milliard de paramètres, et GPT-1 (2018) seulement 117 millions [2, 3, 4].

1.4 Questions de recherche

Ce rapport vise à répondre aux questions suivantes :

- ChatGPT peut-il générer un jeu du Démineur fonctionnel sans instructions détaillées ?
- Est-il plus performant sur des tâches précises que sur des tâches générales ?
- Permet-il réellement un gain de temps pour le développeur ?
- Quels sont ses principaux avantages et limitations ?
- Peut-on envisager que ce type d'outil remplace, à terme, le travail du programmeur ?

2 Méthodes

2.1 Approche expérimentale

Nous avons adopté une approche progressive, en testant ChatGPT sur des tâches de complexité croissante. L'idée est de partir de demandes très générales pour aller vers des demandes de plus en plus spécifiques, afin d'évaluer comment le niveau de précision des instructions influence la qualité du code généré.

2.2 Protocole de test

Les tests ont été réalisés sur la version gratuite de ChatGPT (basée sur GPT-3.5) accessible via l'interface web d'OpenAI. Chaque test a été effectué dans une nouvelle conversation afin d'éviter tout biais lié au contexte accumulé.

2.2.1 Test 1 : Génération complète du jeu

Dans un premier temps, nous avons demandé à ChatGPT de générer un jeu du Démineur complet en Python, sans fournir d'instructions détaillées. Le prompt utilisé était :

> *"Crée un jeu du Démineur en Python."*

L'objectif était d'évaluer la capacité du modèle à produire un programme fonctionnel et jouable à partir d'une demande minimaliste.

2.2.2 Test 2 : Fonctions spécifiques

Nous avons ensuite testé la capacité de ChatGPT à produire des fonctions isolées :

- **Fonction d'affichage** `print_board` : affichage de la grille avec numérotation des lignes et colonnes.
- **Placement des mines** : génération aléatoire des positions des mines.
- **Fonction** `propagate_click` : propagation du clic sur une case vide (version itérative puis récursive).

Pour chaque fonction, nous avons fourni une description précise des attentes, parfois en copiant-collant des extraits de l'énoncé du projet.

2.2.3 Test 3 : Complétion de code

Enfin, nous avons fourni à ChatGPT des parties incomplètes du programme (fonctions manquantes, structure générale) afin d'évaluer sa capacité à *"combler les trous"* de manière cohérente.

2.2.4 Test 4 : Correction de bugs

Nous avons également testé la capacité de ChatGPT à identifier et corriger des erreurs dans du code existant.

2.3 Critères d'évaluation

Les résultats ont été évalués selon les critères suivants :

Critère	Description
Correction syntaxique	Le code s'exécute-t-il sans erreur ?
Cohérence algorithmique	La logique est-elle correcte ?
Jouabilité	Le jeu est-il fonctionnel et jouable ?
Lisibilité	Le code est-il clair et bien structuré ?
Temps de développement	Combien de temps pour obtenir un résultat satisfaisant ?

TABLE 1 – Critères d'évaluation des résultats

3 Résultats

3.1 Test 1 : Génération complète du jeu

Lorsque nous avons demandé à ChatGPT de générer un jeu du Démineur complet sans instructions détaillées, le résultat obtenu était décevant à plusieurs égards :

- **Affichage** : La grille était affichée sous forme d'une simple matrice de nombres, sans bordures, sans numérotation des lignes et colonnes, et sans distinction visuelle entre les cases révélées et non révélées.
- **Initialisation** : La matrice était initialisée avec des valeurs incohérentes (par exemple, des entiers "2" partout).
- **Imports** : Certains modules étaient importés inutilement, tandis que d'autres nécessaires étaient absents.
- **Jouabilité** : Le jeu était techniquement jouable, mais l'expérience utilisateur était très médiocre.

Conclusion partielle : Sans instructions précises, ChatGPT produit un code fonctionnel au sens strict, mais de qualité insuffisante pour un projet académique.

3.2 Test 2 : Fonctions spécifiques

3.2.1 Fonction `print_board`

Lorsque nous avons demandé explicitement une fonction d'affichage utilisant les caractères "-" et "|" pour dessiner la grille, le résultat était nettement meilleur. Cependant, ChatGPT ne gérait pas correctement les décalages d'alignement pour les nombres à deux chiffres.

3.2.2 Placement des mines

Pour cette tâche, ChatGPT a produit un code satisfaisant. Il a spontanément choisi de représenter les positions des mines sous forme de tuples et a correctement utilisé le module `random` pour générer des positions aléatoires sans doublons.

```
1 import random
2
```

```python
def place_mines(rows, cols, num_mines):
    mines = set()
    while len(mines) < num_mines:
        r = random.randint(0, rows - 1)
        c = random.randint(0, cols - 1)
        mines.add((r, c))
    return mines
```

Listing 1 – Code généré pour le placement des mines

3.2.3 Fonction `propagate_click`

Cette fonction est la plus complexe du projet. Elle doit révéler récursivement toutes les cases voisines lorsqu'une case vide (sans mine adjacente) est cliquée.

Version itérative : La première version fournie par ChatGPT contenait plusieurs erreurs :

- Utilisation de variables non définies (`dx`, `dy`).
- Référence à une fonction `get_neighbors()` non implémentée.
- Logique de propagation incorrecte.

Version récursive : Après avoir demandé explicitement une version récursive, ChatGPT a produit un code très proche de la solution attendue, avec une structure claire et une logique correcte.

```python
def propagate_click(board, revealed, row, col, rows, cols):
    if row < 0 or row >= rows or col < 0 or col >= cols:
        return
    if revealed[row][col]:
        return
    revealed[row][col] = True
    if board[row][col] == 0:
        for dr in [-1, 0, 1]:
            for dc in [-1, 0, 1]:
                if dr != 0 or dc != 0:
                    propagate_click(board, revealed, row + dr, col
                        + dc, rows, cols)
```

Listing 2 – Version récursive de `propagate_click`

3.3 Test 3 : Complétion de code

Lorsque nous avons fourni à ChatGPT une structure de code incomplète avec les fonctions `init_game` et `main` partiellement définies, il a réussi à produire une version cohérente et jouable. Bien que superficielle par rapport aux exigences du projet, cette version respectait les règles de base du jeu.

Point positif : ChatGPT a fait preuve d'une bonne "mémoire" contextuelle, conservant la cohérence entre les différentes fonctions au fil de la conversation.

3.4 Test 4 : Correction de bugs

Nous avons soumis à ChatGPT du code contenant des erreurs intentionnelles. Dans la majorité des cas, il a correctement identifié les problèmes et proposé des corrections pertinentes. Toutefois, pour des bugs plus subtils (erreurs de logique, cas limites), ses suggestions étaient parfois incorrectes ou incomplètes.

3.5 Synthèse des résultats

Test	**Syntaxe**	**Logique**	**Jouabilité**	**Qualité globale**
Génération complète	✓	∼	∼	Faible
`print_board`	✓	∼	N/A	Moyenne
Placement mines	✓	✓	N/A	Bonne
`propagate_click` (itératif)	×	×	N/A	Faible
`propagate_click` (récursif)	✓	✓	N/A	Bonne
Complétion de code	✓	✓	✓	Moyenne
Correction de bugs	✓	∼	N/A	Moyenne

TABLE 2 – Synthèse des résultats obtenus

4 Discussion

4.1 Analyse des résultats

Les résultats obtenus confirment notre hypothèse initiale : **ChatGPT est plus performant sur des tâches précises et bien définies que sur des tâches générales et ouvertes**.

Lorsqu'on lui demande de créer un programme complet sans contraintes, il produit un code fonctionnel mais de qualité médiocre, manquant de rigueur dans la présentation et la gestion des cas particuliers. En revanche, lorsqu'on lui fournit des instructions détaillées et des contraintes claires, la qualité du code s'améliore significativement.

4.2 Importance de la formulation des prompts

Un enseignement majeur de cette étude est l'importance cruciale de la **formulation des prompts** (les instructions données au chatbot). Un prompt vague ou ambigu conduit à des résultats décevants, tandis qu'un prompt précis, structuré et contextuel permet d'obtenir des réponses de bien meilleure qualité.

Cette compétence, parfois appelée *"prompt engineering"*, devient de plus en plus importante dans l'utilisation des LLMs. Elle requiert une bonne compréhension du problème à résoudre et une capacité à le décomposer en sous-tâches claires.

4.3 Limites de ChatGPT

Nous avons identifié plusieurs limites importantes :

- **Manque de rigueur** : ChatGPT peut produire du code syntaxiquement correct mais logiquement faux.
- **Hallucinations** : Il lui arrive d'inventer des fonctions ou des modules qui n'existent pas.
- **Absence de tests** : Le code généré n'est jamais testé par le modèle lui-même.
- **Dépendance au contexte** : Sans contexte suffisant, les réponses peuvent être incohérentes.
- **Temps d'interaction** : Obtenir un résultat satisfaisant peut nécessiter de nombreux échanges.

4.4 Avantages de ChatGPT

Malgré ces limites, ChatGPT présente des avantages indéniables :

- **Rapidité** : Il génère du code en quelques secondes.
- **Polyvalence** : Il peut aider sur de nombreux aspects (syntaxe, algorithmes, documentation).
- **Transformation de code** : La conversion itératif/récursif est particulièrement bien gérée.
- **Débogage** : Il peut aider à identifier certaines erreurs.
- **Disponibilité** : Accessible 24h/24, contrairement aux assistants humains.

4.5 ChatGPT : assistant ou remplaçant ?

La question centrale est de savoir si ChatGPT peut remplacer le travail d'un programmeur. Notre étude suggère que **non** : ChatGPT est un outil d'assistance, pas un substitut.

Un programmeur expérimenté peut tirer parti de ChatGPT pour accélérer certaines tâches, mais il doit toujours :

- Vérifier la correction du code généré.
- Comprendre ce que fait le code.
- Adapter le code au contexte spécifique du projet.
- Tester exhaustivement le programme.

Un débutant, en revanche, risque d'utiliser du code qu'il ne comprend pas, ce qui peut être contre-productif pour l'apprentissage et dangereux en termes de qualité logicielle.

5 Conclusion

Ce travail a permis d'évaluer les capacités et les limites de ChatGPT dans le cadre du développement d'un projet informatique. Nos expériences montrent que :

1. ChatGPT est capable de générer du code fonctionnel, mais la qualité dépend fortement de la précision des instructions.
2. Il est plus efficace sur des tâches spécifiques que sur des demandes générales.
3. Son utilisation optimale requiert une expertise préalable en programmation.
4. Il ne remplace pas le programmeur, mais peut constituer un assistant précieux.

L'affirmation selon laquelle les LLMs représentent "l'avenir de la programmation" doit être nuancée. Ces outils transforment indéniablement la manière dont nous développons des logiciels, mais ils ne suppriment pas le besoin de compétences humaines en algorithmique, en conception logicielle et en esprit critique.

En définitive, **ChatGPT est un outil puissant, mais il ne vaut que ce que vaut l'utilisateur qui le manipule**.

Références

Références

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is all you need*. Advances in Neural Information Processing Systems, 30. https://arxiv.org/abs/1706.03762

[2] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving language understanding by generative pre-training*. OpenAI. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[3] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language models are unsupervised multitask learners*. OpenAI Blog, 1(8), 9. https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf

[4] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). *Language models are few-shot learners*. Advances in Neural Information Processing Systems, 33, 1877-1901. https://arxiv.org/abs/2005.14165

[5] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... & Lowe, R. (2022). *Training language models to follow instructions with human feedback*. Advances in Neural Information Processing Systems, 35, 27730-27744. https://arxiv.org/abs/2203.02155

A Annexes

A.1 Architecture du Transformer

Le **Transformer** est une architecture de réseau de neurones introduite en 2017 par Vaswani et al. dans l'article *"Attention Is All You Need"* [1]. Cette architecture a révolutionné le domaine du traitement du langage naturel (NLP) et constitue la base des modèles de langage modernes comme GPT, BERT, T5 et bien d'autres.

A.1.1 Vue d'ensemble

Contrairement aux architectures précédentes basées sur des réseaux de neurones récurrents (RNN) ou des réseaux à convolution (CNN), le Transformer repose entièrement sur le **mécanisme d'attention**. Cette approche permet de traiter tous les mots d'une séquence **simultanément** (en parallèle), plutôt que séquentiellement, ce qui améliore considérablement la vitesse d'entraînement et la capacité à capturer des dépendances à longue distance.

Le Transformer est composé de deux parties principales :

- **Encodeur** (*Encoder*) : Traite la séquence d'entrée et produit une représentation contextuelle riche. Il est utilisé pour comprendre le sens du texte d'entrée.
- **Décodeur** (*Decoder*) : Génère la séquence de sortie mot par mot, en s'appuyant sur la représentation produite par l'encodeur ainsi que sur les mots déjà générés.

Note importante : Les modèles GPT (dont ChatGPT) n'utilisent que la partie **décodeur** du Transformer, car leur tâche est de générer du texte (prédire le mot suivant). En revanche, des modèles comme BERT n'utilisent que l'**encodeur**, car ils sont conçus pour comprendre le texte plutôt que le générer.

A.1.2 Structure d'un bloc Transformer

Chaque bloc (ou couche) du Transformer contient les composants suivants :

1. **Attention multi-têtes** (*Multi-Head Attention*) :
   - C'est le cœur du Transformer.
   - Le mécanisme d'attention permet au modèle de pondérer l'importance de chaque mot par rapport aux autres mots de la séquence.
   - L'attention "multi-têtes" signifie que plusieurs mécanismes d'attention fonctionnent en parallèle, chacun capturant des aspects différents des relations entre les mots.
   - Mathématiquement, l'attention est calculée comme suit :

     $$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

     où Q (Query), K (Key) et V (Value) sont des projections linéaires de l'entrée, et d_k est la dimension des clés.

2. **Réseau feed-forward** (*Feed-Forward Network*) :
   — Après l'attention, chaque position passe par un réseau de neurones à deux couches avec une activation non linéaire (généralement ReLU ou GELU).
   — Ce réseau est appliqué indépendamment à chaque position.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

3. **Connexions résiduelles** (*Residual Connections*) :
   — Autour de chaque sous-couche (attention et feed-forward), une connexion résiduelle additionne l'entrée à la sortie.
   — Cela facilite l'entraînement de réseaux profonds en permettant au gradient de circuler plus facilement.

$$\text{Output} = \text{LayerNorm}(x + \text{Sublayer}(x))$$

4. **Normalisation de couche** (*Layer Normalization*) :
   — Appliquée après chaque sous-couche pour stabiliser l'entraînement.
   — Normalise les activations sur la dimension des caractéristiques.

A.1.3 Encodage positionnel

Puisque le Transformer traite tous les mots en parallèle, il n'a pas de notion intrinsèque de l'ordre des mots. Pour résoudre ce problème, un **encodage positionnel** (*Positional Encoding*) est ajouté aux embeddings d'entrée.

Cet encodage utilise des fonctions sinusoïdales de différentes fréquences :

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

où pos est la position du mot dans la séquence, i est la dimension, et d est la dimension totale de l'embedding.

A.1.4 Schéma de l'architecture

La figure ci-dessous illustre l'architecture complète du Transformer :

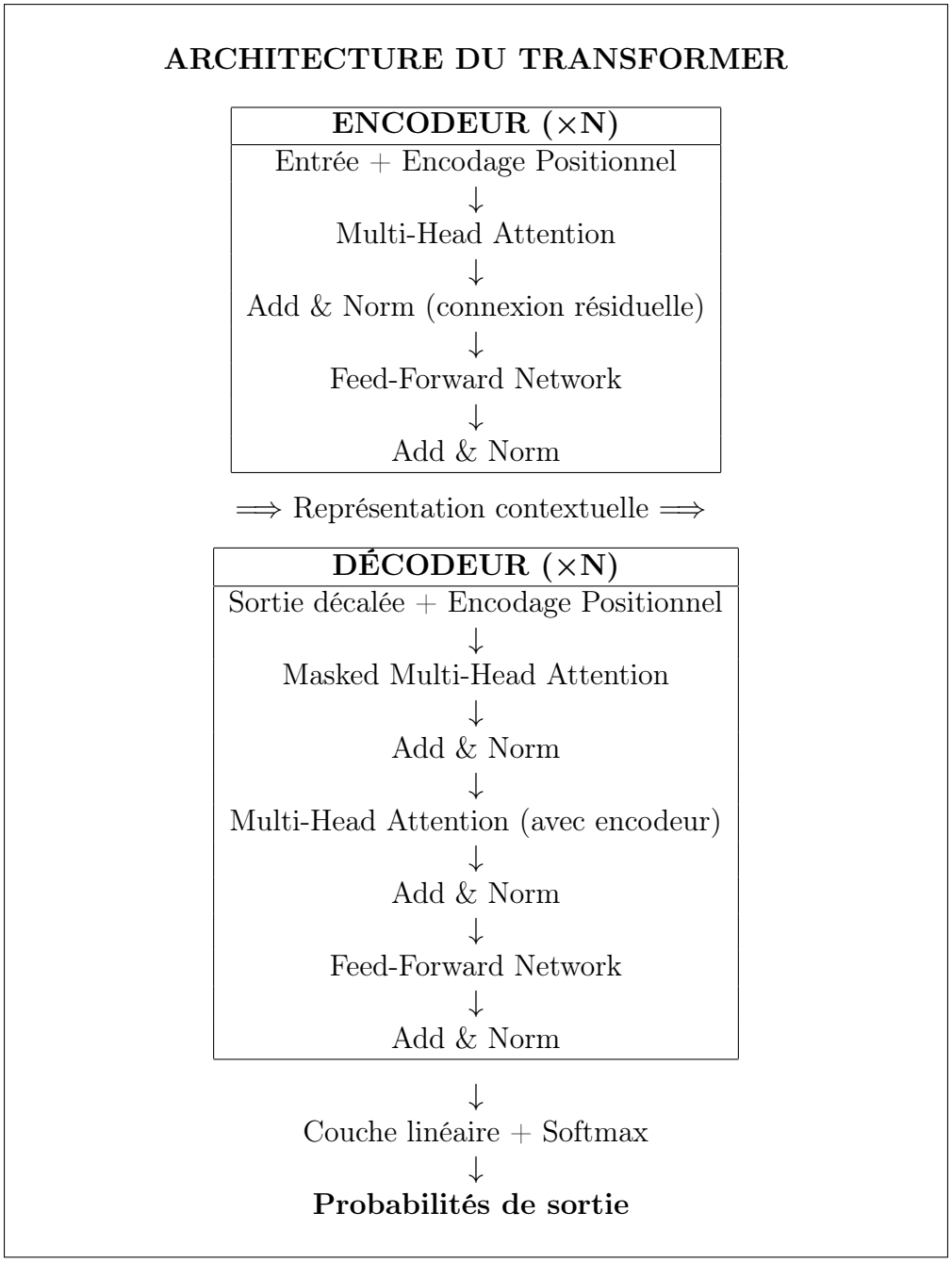

FIGURE 1 – Architecture simplifiée du Transformer (Vaswani et al., 2017)

A.1.5 Paramètres clés

Les modèles Transformer sont caractérisés par plusieurs hyperparamètres :

Paramètre	Description	Valeur typique
N	Nombre de couches (blocs)	6 à 96
d_{model}	Dimension des embeddings	512 à 12288
h	Nombre de têtes d'attention	8 à 96
d_{ff}	Dimension du feed-forward	2048 à 49152
Vocabulaire	Nombre de tokens	30000 à 100000

TABLE 3 – Hyperparamètres typiques des modèles Transformer

A.1.6 Évolution : de GPT-1 à GPT-4

Modèle	Année	Paramètres	Données d'entraînement
GPT-1	2018	117 millions	$\sim \$5 Godetexte$
GPT-2	2019	1,5 milliard	$\sim \$40 Godetexte$
GPT-3	2020	175 milliards	$\sim \$570 Godetexte$
GPT-3.5 (ChatGPT)	2022	$\sim \$175 milliards$	Non divulgué
GPT-4	2023	Non divulgué	Non divulgué

TABLE 4 – Évolution des modèles GPT

A.1.7 Coût d'entraînement

L'entraînement de grands modèles de langage nécessite des ressources computationnelles considérables. Selon une étude de 2020, entraîner un modèle de 1,5 milliard de paramètres (comme GPT-2) coûte environ **1,6 million de dollars** en ressources cloud.

Pour GPT-3 (175 milliards de paramètres), le coût estimé d'entraînement est de l'ordre de **4 à 12 millions de dollars**, sans compter les coûts de recherche et développement.

A.1.8 Résumé

Le Transformer est une architecture révolutionnaire qui a permis le développement des LLMs modernes. Ses caractéristiques clés sont :

— **Parallélisation** : Traitement simultané de tous les tokens.
— **Attention** : Capture des relations à longue distance.
— **Scalabilité** : Capacité à augmenter le nombre de paramètres.
— **Polyvalence** : Applicable à de nombreuses tâches NLP.

C'est cette architecture qui permet à ChatGPT de générer du texte cohérent, de comprendre le contexte d'une conversation et de produire du code informatique.