Les modèles de langage pour la génération de code tiennent-ils leurs promesses ?

Gaël de Chalendar, Pauline Auda, Jérôme Deshayes-Chossart, Olivier Ferret,
Patrick Hède, Hervé Le Borgne, Adolphe Ngosso Ebene,
Ansgar Radermacher, Julien Tourille
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
gael.de-chalendar@cea.fr, firstname.name@cea.fr

RÉSUMÉ

Les modèles de génération de code obtiennent (pour la plupart) les scores qu'ils prétendent obtenir. De nombreux articles, validés par les pairs ou non, publient des scores ainsi que ceux des modèles précédents. Il n'est pas toujours facile de savoir lesquels ont été reproduits par les auteurs ou simplement copiés. De même, les détails du matériel utilisé et autres paramètres ne sont pas toujours disponibles. Nous avons mesuré les performances de plusieurs modèles de l'état de l'art sur plusieurs benchmarks, sur notre propre infrastructure informatique. Nous constatons que nous sommes globalement capables d'obtenir des résultats proches de ceux publiés.

ABSTRACT

Do LLMs for code generation live up to their promises?

Code generation models (mostly) get the scores they claim. Many papers, peer-reviewed or not, publish their own scores as well as those of previous models. It's not always easy to know which have been reproduced or simply copied. Similarly, details of the hardware used and other parameters are not always available. We have measured the performance of several state-of-the-art models on several benchmarks, on our own IT infrastructure. We have found that, overall, we are able to achieve results close to those published.

MOTS-CLÉS : Génération de code, évaluation, reproduction, épistémologie, LLM.

KEYWORDS: Code generation, evaluation, reproduction, epistemology, LLM.

1 Introduction

L'essor des modèles de fondation (FM) et des grands modèles de langage (LLM) a mené à des succès remarquables dans l'*Intelligence Artificielle (IA) générative*. Le lancement de ChatGPT en novembre 2022 a significativement accru le public intéressé par le sujet et le nombre d'applications construites sur de tels modèles a augmenté simultanément. Parmi celles-ci, la *génération de code* permet d'automatiser des tâches de programmation et ainsi augmenter significativement la productivité des programmeurs tout en les laissant se concentrer sur les aspects les plus riches de leur métier. Des modèles sont publiés quotidiennement, avec des performances rapportées toujours croissantes, mais il n'est pas toujours aisé de déterminer précisément le contexte dans lequel les résultats sont obtenus et si les chiffres rapportés sont des scores obtenus par les auteurs ou rassemblés à partir d'autres sources.

Au-delà d'une revue de la littérature, nous réalisons dans cet article une large évaluation comparative des modèles d'IA générative pour la génération de code. Contrairement à des revues comme (Zheng *et al.*, 2024), nous ne nous contentons pas de rapporter les affirmations des auteurs mais reproduisons leurs résultats. Nous effectuons cet effort de reproduction de grande envergure dans un cadre contrôlé, en exploitant notre propre infrastructure informatique.

Dans la suite, nous justifions l'intérêt d'un tel effort de reproduction (section 2) puis décrivons l'organisation de notre évaluation (section 3). Nous présentons ensuite les modèles (section 4), les ensembles de données (section 5) et les benchmarks (section 6) que nous avons étudiés. Enfin, la section 7 présente et discute tous les résultats que nous avons obtenus, avant de conclure.

2 Reproduire : une perspective épistémologique

Comme identifié par Fidler & Wilcox (2021), la reproductibilité est une caractéristique distinctive de la science mais demeure malheureusement en pratique bien plus un principe exprimé qu'une réalité factuelle. Cette insuffisance est souvent désignée par le terme de « crise de la réplication ». La principale cause est davantage la structure des incitations dans la science que la fraude ou les pratiques de recherche dites douteuses. Fidler & Wilcox (2021) se concentrent principalement sur les sciences sociales (et en particulier sur la psychologie) et les sciences de la vie. Ils rapportent le cas, en psychologie, de l'article de Bem (2011) *prouvant* que la perception extrasensorielle existe et qui a été accepté dans une revue, tandis que quatre autres articles montrant des échecs de reproduction de ce travail ont été rejetés. Ils font également état, en médecine, des travaux de Ioannidis (2005), montrant qu'il y a trop de faux positifs avec une puissance statistique insuffisante. Les projets de reproduction en psychologie et en médecine rapportent un échec de reproduction dans environ 50 % des cas. « Bien que la réplication soit souvent considérée de manière désinvolte comme une pierre angulaire de la méthode scientifique, les études de réplication directe [...] sont un événement rare dans la littérature publiée [...]. » Ce constat est en partie lié aux politiques des éditeurs favorisant la nouveauté. Il y a par ailleurs une forte focalisation sur la significativité statistique, mais une ignorance de la puissance statistique. En utilisant une expérience de pensée bayésienne, Fidler & Wilcox (2021) montrent que la réplication n'est pas concluante, mais informative tout de même. Enfin, ils affirment que les méthodes et les matériaux ouverts sont le modèle le plus efficace. En Traitement Automatique des Langues (TAL), les tâches partagées sont utilisées au moins depuis les années quatre-vingt-dix (Harman, 1993) pour comparer les résultats de différents systèmes sur les mêmes données. L'objectif n'était toutefois pas la reproduction, les méthodes d'évaluation et les sujets évoluant constamment avec les évolutions des technologies. Plus récemment, l'intérêt pour la reproductibilité a augmenté avec REPROLANG2020 (Branco *et al.*, 2020) et les ateliers ReproGen initiés par Anya Belz (Belz *et al.*, 2021b, 2022), suite à sa revue de la recherche sur la reproductibilité en TAL (Belz *et al.*, 2021a) où ses collègues et elle-même ont conclu que très peu de reproductions obtenaient des résultats très similaires et que de petites variations avaient beaucoup d'impact.

Dans un commentaire au journal EMBO, Dirnagl (2019) calcule que les recherches très prospectives et nouvelles, en science médicale, produiront naturellement environ 40% de faux positifs. Il complète en affirmant que ce n'est pas un problème si les résultats principaux sont confirmés par des reproductions correctes et conclut par « Nous devons rétablir la reproductibilité à sa position de pierre angulaire de la science, conceptuellement inséparable de l'exploration. »

3 Mise en place de l'évaluation comparative

Pour réaliser ce benchmark, nous avons testé uniquement des modèles pouvant être déployés et exécutés gratuitement sur notre propre infrastructure, excluant ainsi des modèles comme OpenAI GPT ou Google Gemini. L'utilisation des codes et modèles bruts étant très difficile, nous avons exploité autant que possible Big Code Evaluation Harness [1] (BCEH), qui permet de générer du code pour plusieurs benchmarks avec tout modèle autorégressif disponible sur le hub HuggingFace. BCEH fournit également des images Docker pour exécuter les tâches d'évaluation en toute sécurité. En pratique, notre évaluation comparative des modèles de génération de code a consisté à maintenir une liste de modèles et technologies, lister les méthodes d'évaluation et les corpus associés. À partir de cela nous avons choisi les tâches à reproduire, déployé les modèles, déployé les benchmarks et les avons exécutés.

Les sections suivantes couvrent ces tâches. Pour la génération et l'évaluation, nous avons travaillé sur un cluster de calcul à échelle pétaflopique en évolution régulière, actuellement composé de 40 nœuds de calcul, représentant 257 GPU, avec une majorité de NVIDIA A100.

4 Modèles

Il existe un très grand nombre de LLM capables de générer du code, soit généralistes mais incluant du code dans leur corpus de pré-entraînement, soit spécialement fine-tunés ou autrement spécialisés pour le code. De plus, de nouveaux modèles apparaissent chaque jour. Nous avons commencé cette évaluation comparative en juin 2023 et arrêté d'ajouter de nouveaux modèles en février 2024. Dans cette section, nous décrivons de nombreux modèles, ceux que nous avons évalués mais également d'autres. Particulièrement, nous avons décidé de n'évaluer que des modèles open source ou au moins avec des poids ouverts que nous pouvions exécuter sur notre propre matériel, excluant ainsi des modèles disponibles uniquement via des API basées sur le cloud, comme les modèles OpenAI. Nous décrivons néanmoins certains modèles fermés dans la session actuelle pour montrer les spécificités du plus grand nombre de modèles possibles.

Le tableau 2 répertorie les modèles et leurs principales caractéristiques. Nous avons commencé par compiler les informations présentées par Zheng *et al.* (2023). Nous les avons complétées et mises à jour avec des informations provenant de diverses sources. Il manque certaines entrées, *a priori* intéressantes (SantaCoder, CodeT5+...), mais il est impossible de couvrir l'ensemble des modèles de codage ici. La page Open LLMs donne sur tous types de LLM des informations supplémentaires sur les modèles, les benchmarks, les ensembles de données, etc. Pour chaque modèle, le tableau 2 donne plusieurs informations : si le modèle est ouvert et comment ; s'il supporte plusieurs langages de programmation ; sa taille en nombre de paramètres ; des informations sur le ou les jeux de données utilisés pour l'entraîner (son origine, les langages de programmation qu'il contient, la taille en nombre de tokens de texte et de code qu'il contient) ; et enfin le type d'évaluations qu'il rapporte (sont-elles multilingues et les noms des benchmarks). Il apparaît que très peu de modèles, et particulièrement parmi les plus récents, sont vraiment libres (Gratis vs libre, 2024). Lorsque les modèles sont disponibles gratuitement, il y a souvent des restrictions à leur utilisation, comme DeepSeek interdisant toute utilisation militaire par exemple. D'autres interdisent l'utilisation commerciale (Falcon), limitent le nombre d'utilisateurs commerciaux (Llama2) ou permettent uniquement les activités de recherche

1. https://github.com/bigcode-project/bigcode-evaluation-harness?search=1

(CodeGeeX). Certains modèles ont des poids complètement libres mais leur code d'entraînement ou leur corpus ne le sont pas (CodeGen2, Mistral…). Finalement, dans ce tableau, le seul modèle vraiment libre et récent est StarCoder, ainsi que sa version plus récente StarCoder2.

Dans le reste de cette section, nous donnons plus d'informations sur une sélection des entrées du tableau 2. Nous décrivons également quelques autres modèles qui n'ont pas été inclus dans le tableau.

CodeBERT (Feng *et al.*, 2020) est une application précoce des Transformers à la génération de code. C'est un BERT (Devlin *et al.*, 2019) entraîné avec du code dans ses sorties. Il n'a pas été testé sur la génération de code, mais seulement sur la recherche de code et la génération de documentation. C'est pourquoi il n'a pas été inclus dans notre benchmark. CodeT5 (Wang *et al.*, 2021) est un modèle encodeur-décodeur. Sa phase de pré-entraînement comprend une procédure de masquage utilisant des identifiants de sorte que le modèle puisse retrouver leur valeur lorsque cela est nécessaire. Il a été testé sur plusieurs tâches. En génération de code, il a été évalué sur les données de Concode (Iyer *et al.*, 2018) avec Exact Match (EM), BLEU et CodeBLEU. Il n'a pas été évalué en utilisant des benchmarks exécutant le code généré et vérifiant ses capacités. Llama2 est un LLM proposé par Meta. Il est gratuit avec une limitation sur le nombre d'utilisateurs mensuels s'il est intégré dans une application commerciale. Un certain nombre de travaux ont été initiés à partir de Llama-2 [2], notamment un fine-tuning pour la génération de Python [3] et un fine-tuning efficace en paramètres pour plusieurs langages de programmation [4]. Les modèles CodeGeeX et CodeGeeX2 (Zheng *et al.*, 2023) utilisent une architecture de décodage GPT (generative pre-training). Ils ont été pré-entraînés sur de grands corpus de code avec plusieurs langages de programmation. Ils sont libres d'utilisation en inférence uniquement pour la recherche, avec des restrictions interdisant toute utilisation militaire ou portant atteinte à la sécurité nationale et à l'unité de la Chine. Pour une utilisation commerciale, une licence spécifique doit être demandée. Mais le code, y compris pour l'entraînement, est vraiment libre, sous licence MIT. Avec leurs modèles, les auteurs ont publié le benchmark HumanEval-X (cf. section 6) afin de pouvoir vérifier la capacité des modèles à traiter plusieurs langages de programmation. BLOOM (Scao *et al.*, 2023) est un modèle de langage généraliste de type décodeur uniquement (176 mds de paramètres) pré-entraîné sur 46 langues naturelles et 13 langages de programmation. Son principal intérêt est qu'il a été publié, y compris le code et le modèle, avec une licence vraiment libre. En tant qu'initiative dirigée par HuggingFace, il a également été une source d'inspiration pour StarCoder. En tant que modèle généraliste non fine-tuné ou spécifiquement préparé pour la génération de code, BLOOM n'a pas présenté des performances particulièrement bonnes sur la génération de code, avec une valeur pass@1 de 15,5, un peu au-dessus des 11,6 rapportés de GPT-J ; mais GPT-J n'a que 6 mds de paramètres, comparé aux 176 mds de BLOOM. Nous aurions pu penser qu'il y aurait eu de nombreux fine-tunings de BLOOM mais il a été supplanté par des modèles spécialisés, comme StarCoder, et nous n'avons pas pu trouver de tels exemples. Nous avons décidé de ne pas inclure BLOOM dans notre benchmarking. Lemur (Xu *et al.*, 2023) est un fine-tuning de Llama2, héritant donc de son statut non libre. GraphCodeBERT (Guo *et al.*, 2020) est un modèle intéressant qui exploite la structure du code (mais n'encode pas l'AST). Malheureusement, il n'a pas été utilisé pour la génération de code mais sur quatre autres tâches (recherche de code, détection de clones, traduction de code et raffinement de code). De ce fait, il n'a pas été évalué ici.

2. https://github.com/facebookresearch/llama
3. https://pub.towardsai.net/fine-tuning-a-llama-2-7b-model-for-python-code-generation-865453afdf73
4. https://github.com/juyongjiang/CodeUp

	Propriétés des Modèles			Jeux de Données			Évaluation	
	Open	Multi langages	#Params	Source	Langages	Tailles	Éval. Multi-lang.	Benchmarks
Codex (Chen *et al.*, 2021)	N	N	12B	Collecté	Python	Code : 159GB	N	HumanEval, APPS
GPT-Neo (Black *et al.*, 2021)	O	O	1.3B, 2.7B	The Pile	Multiple	Texte : 730GB Code : 96GB (400B tokens trained)	N	HumanEval
GPT-J (Wang & Komatsuzaki, 2021)	O	O	6B	The Pile	Multiple	Texte : 730GB Code : 96GB (473B tokens entraînés)	N	HumanEval
CodeGeeX2 (Zheng *et al.*, 2023)	IRS	O	6B	The Pile, CodeParrot, Collecté	>100	CodeGeeX+600B	O	HumanEval, HumanEval-X DS1000
StarCoder (Li *et al.*, 2023)	O	O	15.5B	The Stack v1.2	>80	1 Milliard	Y	HumanEval-X, HumanEval MBPP, MultiPL-E
CodeGen2 (Nijkamp *et al.*, 2023a)	I	O	7B	StarCoderData	86	Code : 250BT		HumanEval, HumanEval-Infill, XSum, CodeXGLUE, SuperGLUE
Llama2 (Rozière *et al.*, 2023)	NL	O	34B	« code disp. publiquement »	>= 7	500BT (from Llama2)	O	HumanEval, MBPP, APPS, MultiPL-E
Code Llama (Rozière *et al.*, 2023)	NL	O	34B	« code disp. publiquement »	>= 7	500BT (from Llama2)	O	HumanEval, MBPP, APPS, MultiPL-E
Mistral7B (Jiang *et al.*, 2023)	I	O	7B	NP	NP	NP	N	HumanEval, MBPP
ChatGPT-4 (OpenAI *et al.*, 2023)	N	O	NP	NP	NP	NP	O	HumanEval

TABLE 2 – LLM préentraînés de l'état de l'art relatifs à des langages de programmation. Données initiales issues de (Zheng *et al.*, 2023). Tableau complet en annexe.

Ouverture : IR=inférence seulement avec restrictions ; IRS=inférence recherche seulement (tout le reste sous licence commerciale et source fermée ; NC=Non commercial ; I=inférence ouverte/libre, pas d'information sur l'entraînement ; NL=Ouvert mais non libre. Tous : NP=Non publié/Inconnu

5 Jeux de données

Le tableau 3 liste les jeux de données utilisés par les modèles étudiés. Dans ce qui suit, nous donnons des détails supplémentaires sur chacun d'eux.

Concode est un jeu de données conçu pour les tâches de génération de code. Il se compose de problèmes de programmation en Java et de leurs solutions correspondantes, extraites de bases de code et de documentation. Le jeu de données est utilisé pour entraîner des modèles à générer des extraits de code à partir de descriptions en langage naturel. Il ne s'agit pas d'un grand corpus de pré-entraînement, mais d'un corpus destiné à l'entraînement direct du modèle de type LSTM correspondant (Iyer *et al.*, 2018).

The Pile est un jeu de données de 825 Go de textes divers, compilé par EleutherAI. Il inclut du contenu provenant de diverses sources telles que des livres, Wikipédia, des articles académiques et des pages web. Le jeu de données vise à supporter l'entraînement de modèles de langage de grande taille sur une large gamme de sujets et de styles. Il est principalement composé de textes en anglais et bien qu'il ait été librement disponible, il ne l'est plus en raison de problèmes de droits d'auteur.

The Stack est un vaste jeu de données de code sous licence permissive, compilé pour soutenir la recherche en génération et compréhension de code. Il inclut du code source provenant de nombreux langages de programmation et référentiels, en se concentrant sur des exemples de code de haute qualité et utilisables. Il a été collecté par le projet BigCode afin de soutenir la création de StarCoder. Il

	Langages	Caractéristiques	Statut
Concode (Iyer *et al.*, 2018)	Java	Comment.+Java code+Contexte	100k exemples entr., 2k valid./test
The Pile (Gao *et al.*, 2020)	97% Anglais	800 GB	Indisponible (copyright)
The Stack (Kocetkov *et al.*, 2022)	30	3.1TB	Disponible ici : https://huggingface.co/bigcode
ROOTS (Laurençon *et al.*, 2022)	46 LN, 13 Code	1.6TB	Pas de détails sur les tailles des sous-parties
RefinedWeb (Penedo *et al.*, 2023)	176 LN	5,000GT (600GT public)	Pas d'info. sur les quantités de code Disponible ici : https://huggingface.co/datasets/tiiuae/falcon-refinedweb
Code Contests (Li *et al.*, 2022)	12 pré-entraînement, Solutions Python et Java	715GB pré-entraînement, 13k problèmes >10test/pb	Disponible ici : https://github.com/google-deepmind/code_contests

TABLE 3 – Jeux de données pour la génération de code.

est maintenant supplanté par The Stack v2, créé pour StarCoder2. Il est dérivé de Software Heritage [5], la plus grande archive publique de code source.

ROOTS est un jeu de données multilingue et multi-domaines conçu pour entraîner des modèles de langage de grande taille avec des textes divers du monde entier. Il couvre une large gamme de langues et de domaines, tels que la littérature, les actualités, les médias sociaux et les langages de programmation, visant à créer des modèles d'IA inclusifs et complets. C'est le corpus qui a été utilisé pour entraîner BLOOM.

RefinedWeb (Penedo *et al.*, 2023) est un jeu de données de texte web de haute qualité, préparé pour entraîner des modèles de langage à grande échelle de la famille Falcon. Il implique le filtrage et le nettoyage des pages web pour éliminer le contenu de faible qualité, en s'assurant que le texte restant est adapté à un entraînement IA robuste. Il a été construit à partir des Common Crawls et est publié sous la licence permissive Open Data Commons Attribution License (ODC-By) v1.0 [6].

Code Contests est composé de défis de programmation et de solutions, généralement issus de Codeforces, Description2Code et CodeNet. Il a été utilisé pour affiner le modèle AlphaCode de Deepmind (Li *et al.*, 2022).

6 Benchmarks

Les benchmarks évaluent les modèles grâce à des mesures de performance. Les premiers sur la génération de code utilisaient des mesures de similarité avec des programmes de référence à base de compte de n-grammes, issues des mesures en TAL, ROUGE pour le résumé automatique et BLEU pour la traduction automatique. Mais ces mesures, d'une part n'exploitent pas le fait que les langages de programmation utilisent une grammaire formelle et d'autre part, jugent négativement des programmes fonctionnellement identiques mais écrits différemment. Pour le premier problème, des mesures adaptées au code ont été développées, comme CodeBLEU (Ren *et al.*, 2020). Mais les benchmarks récents utilisent plutôt $pass@k$, une métrique qui évalue la probabilité, en examinant k résultats générés, qu'au moins l'un d'eux soit valide, généralement en s'assurant qu'il passe un ensemble de tests unitaires (Chen *et al.*, 2021). Le tableau 4 liste les benchmarks que nous avons utilisés avec

5. https://www.softwareheritage.org/
6. https://opendatacommons.org/licenses/by/1-0/

certaines de leurs caractéristiques et les sous-sections ci-dessous les décrivent plus précisément. Il faut noter que la partie textuelle de tous ces benchmarks, les requêtes, est systématiquement en anglais. Une table plus complète de benchmarks est donnée en annexe.

	Langages	Caractéristiques	Nb problèmes
HumanEval (Chen *et al.*, 2021)	Python	Prompt LN+tests unitaires	164
MBPP Mostly Basic Programming Problems (Austin *et al.*, 2021)	Python	problèmes de programmes avec tests et exemples de solutions	974
MultiPL-E (Cassano *et al.*, 2023)	19	traduction de HumanEval et MBPP vers d'autres langages par compilateurs	164 + 974

TABLE 4 – Benchmarks pour la génération de code. Tableau complet en annexe.

HumanEval est à notre connaissance le premier benchmark évaluant les programmes générés par leur capacité à être exécutés et à passer des tests unitaires. Il contient 164 descriptions de tâches à réaliser ainsi que l'entête de la fonction python à générer. Chaque tâche est accompagnée de plusieurs tests unitaires que le programme doit réussir pour être validé. De nombreux autres benchmarks en ont été dérivés (cf. section B des annexes). MTPB (Multi-Turn Programming Benchmark) (Nijkamp *et al.*, 2023b) contient 115 problèmes écrits par des experts, chacun comprenant une description multi-étapes en langage naturel (invite). Pour résoudre un problème, un modèle doit synthétiser des sous-programmes fonctionnellement corrects (1) suivant la description à l'étape actuelle et (2) en tenant compte des descriptions et des sous-programmes synthétisés lors des étapes précédentes. MBPP (Mostly Basic Programming Problems) (Austin *et al.*, 2021) est composé de 974 courts programmes Python construits par crowd-sourcing auprès de personnes ayant des connaissances de base en Python. Il a été demandé aux participants d'écrire un court énoncé de problème, une seule fonction Python autonome répondant au problème spécifié et trois cas de test vérifiant la correction sémantique de la fonction. Le jeu de données Multipl-E (Cassano *et al.*, 2022) consiste en une traduction automatique des jeux de données HumanEval et MBPP vers d'autres langages. Il supporte 18 langages de programmation.

7 Résultats

Cette section décrit les résultats que nous avons obtenus pour les modèles et benchmarks que nous avons testés [7]. Nous avons évalué 21 modèles avec HumanEval, 6 avec MBPP et 3 avec Multipl-E. Exécuter du code inconnu, et ici généré par IA, est dangereux. Toutes les exécutions ont été effectuées dans des conteneurs sans possibilité d'effet de bord en dehors de ceux-ci. L'exécution des tests multi-langages est coûteuse en temps et en puissance de traitement car chaque test doit être exécuté pour chaque langage. Les tableaux 5, 6 et 7 montrent les résultats obtenus, respectivement sur HumanEval, Multiple-E et MBPP.

Au début du projet, nous calculions *pass@1*, *pass@10* et *pass@100*, mais les utilisateurs ne sont intéressés que par l'obtention d'une seule réponse et éventuellement par des discussions pour la corriger. En outre, le taux de réussite des derniers modèles est si élevé que les autres réponses ne sont probablement plus très utiles. C'est pourquoi, à l'instar d'autres articles récents (Lozhkov *et al.*, 2024), nous n'indiquons ici que la mesure *pass@1*.

7. Les résultats complets sont en annexe.

Modèle	Publié	Reproduit
DeepSeek-Base-1.3B	34,8	33,5
DeepSeek-Instruct-33B	79,3	61,3
DeepSeek-Instruct-6.7B	–	71,3
Llama2-13B	18,3	6,7
Llama2-70B	29,9	20,6
CodeLlama-34B	48,8	47,0
Phind-CodeLlama-34B-v2	73,6	72,7
Mistral 7B	30,5	6,3
StarCoder-15B	33,6	33,8
StarCoder2-3B	31,7	30,8
StarCoder2-7B	35,4	34,6
StarCoder2-15B	46,3	44,7
Dolphin:starcoder2-15B	–	54,5

TABLE 5 – Quelques résultats sur HumanEval. Tableau complet en annexe.

On voit, avec les valeurs obtenues pour HumanEval dans le tableau 5, que la performance reproduite est en général assez proche de la performance rapportée. C'est notamment le cas pour CodeLlama et les variantes de StarCoder. Il y a cependant quelques exceptions : la performance reproduite de Llama2 est beaucoup plus faible que celle rapportée. De même pour Mistral. De manière surprenante, nous avons reproduit une performance plus faible pour le modèle DeepSeek 33B par rapport au modèle plus petit (7B). Ce n'était pas le cas pour les chiffres rapportés. Les variantes plus grandes d'un même modèle ont des performances plus élevées que les variantes plus petites, comme attendu. Par exemple, les performances de StarCoder2 avec des tailles de 3B, 7B et 15B ont été reproduites dans une fourchette allant de 30,8 % à 44,7 %. Les résultats de StarCoder2-7B sont légèrement meilleurs que ceux de StarCoder-15B, deux fois plus grand. Mais l'entraînement avec des instructions peut compenser la taille, les LLM instruits ayant une performance considérablement meilleure que leurs homologues non instruits.

La figure 1 représente les scores de chaque modèle sur HumanEval en fonction de la taille du modèle. Chaque couleur représente un modèle, les carrés correspondant aux scores rapportés et les cercles, aux scores reproduits. Nous aurions aimé avoir des valeurs pour plusieurs tailles de chaque modèle mais avec les données actuelles, il est déjà clair que les scores progressent avec la taille du modèle et avec des modèles plus spécialisés (affinés, instruits…), comme indiqué dans la littérature. Le graphique montre clairement que les scores de Llama2 (en olive), qui n'est pas spécialisé sur le code, sont assez faibles par rapport aux modèles spécialisés de taille similaire. Au contraire, le modèle spécialisé CodeLlama (en bleu), entraîné sur le code après initialisation à partir des poids de Llama2, et encore plus Phind-CodeLlama (en rouge), une version instruite de CodeLlama, obtiennent des résultats très élevés. Les résultats de StarCoder2 (noir) sont les meilleurs pour leur taille.

Le tableau 6 montre un extrait des résultats du benchmark MultiPL-E exécuté avec trois modèles différents : un DeepSeekCoder instruit et deux variantes de StarCoder2. Nos résultats pour le langage Go sont proches de 0, contrairement aux chiffres rapportés par les données de référence, qui sont d'environ 20 %. Cela pourrait indiquer un problème avec notre exécution du test. Le modèle StarCoder2-7B obtient des résultats nettement meilleurs que StarCoder-15B bien qu'il soit plus petit. Nous pouvons observer que les modèles obtiennent de meilleurs résultats pour des langages populaires, tels que

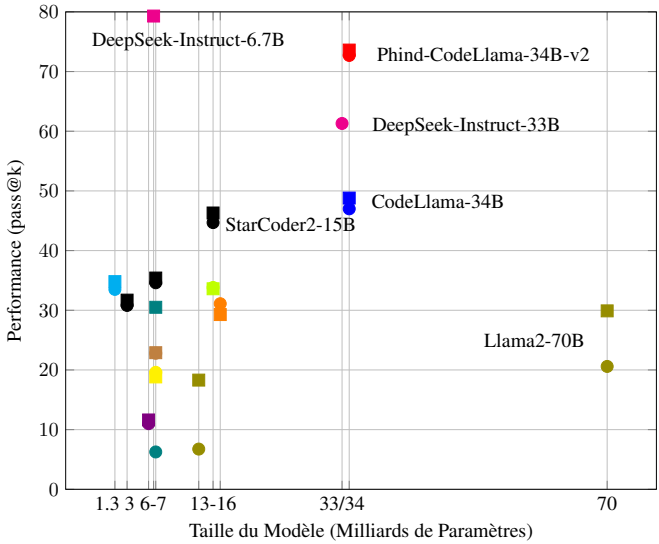

FIGURE 1 – Performances sur HumanEval
Bleu : CodeLlama, Orange : CodeGen, Jaune : CodeGen2 Magicode-OSS-Instruct, Vert : Dolphin StarCoder2, Magenta : DeepSeek-Instruct, Violet : GPT-J, Olive : Llama2

Python et JavaScript, comparés à d'autres, moins répandus, comme par exemple D.

Le tableau 7 montre les résultats du benchmark MBPP. StarCoder2-15B a les meilleurs résultats avec 56,7 %, bien que CodeLlama-34B, qui est légèrement derrière (55 %), soit plus gros. Il est à noter que nous n'avons pas pu reproduire le score rapporté de StarCoder2-7B (54,40 %), n'obtenant que 38,48. En fait, nous avons reproduit un score similaire seulement avec StarCoder2-15B (56,74 %).

8 Limitations

En testant différents modèles, nous avons constaté certaines divergences entre les performances revendiquées et les résultats réels. Dans certains modèles, nous avons observé que les développeurs utilisaient des techniques de post-traitement pour s'assurer que les modèles passent les tests. Par exemple, dans le modèle Codegeex, ou dans BigCode Evaluation Harness, des imports personnalisés ont été ajoutés après la génération du code comme post-traitements. Plus important encore, le code est coupé après la première fonction pour passer le test d'évaluation HumanEval (Github, 2023). Cela soulève la question de savoir si appliquer de tels post-traitements permet une comparaison équitable par rapport aux modèles ajoutant indépendamment les imports et le jeton "eos" (fin de séquence) au bon endroit.

Une information importante que nous n'avons pas collectée est le temps de traitement pour chaque modèle. De futures exécutions devraient inclure cette information. De même, nous n'avons pas abordé dans cette étude les questions éthiques et pratiques liées à l'utilisation des outils de génération de code. Ces questions sont en réflexion dans l'ensemble du domaine de l'IA générative (voir par exemple

Langage	DeepSeek-Instruct-33B		StarCoder-15B		StarCoder2-7B	
	Publié	Reproduit	Publié	Reproduit	Publié	Reproduit
C++	68,9	62,6	31,6	30,4	33,6	40,2
D	–	–	13,6	13,0	15,1	16,5
Go	–	0,0	17,6	0,0	20,2	0,6
Java	73,4	51,3	30,2	28,2	29,4	35,5
JavaScript	73,9	62,4	30,8	30,9	35,4	35,9
R	–	39,4	15,5	16,1	16,7	21,4
Ruby	–	45,1	1,2	1,6	28,3	28,6
Racket	–	26,1	0,1	10,0	11,6	12,9
Rust	–	49,1	21,8	22,1	29,6	29,2
Bash	43,0	–	10,5	11,4	12,2	14,9

TABLE 6 – Résultats sur MultiPL-E pour quelques langages. Tableau complet en annexe.

Modèle	MBPP pass@1	
	Publié	Reproduit
CodeLlama-34B	55,0	55,0
StarCoder	52,7	43,2
Mixtral-8x7B-Instruct	–	45,8
StarCoder2-3B	–	37,6
StarCoder2-7B	54,4	38,5
StarCoder2-15B	–	56,7

TABLE 7 – Résultats sur MBPP

Zohny *et al.* (2023)) et devront être étudiées pour la génération de code. Enfin, les benchmarks de nos expérimentations se concentrent sur l'anglais du point de vue des langues utilisables pour les prompts. Le tout récent benchmark HumanEval-XL (Peng *et al.*, 2024) élargit le spectre des langues pouvant être testées et serait intéressant de ce point de vue à ajouter à l'ensemble des benchmarks que nous avons considérés.

9 Conclusion et perspectives

Dans cet article, nous avons présenté les résultats d'une expérience de reproductibilité concernant les modèles de génération de code. Plus précisément, nous avons exécuté plusieurs benchmarks sur de multiples modèles de génération de code open source. Nous avons principalement réussi à reproduire les résultats publiés sur notre matériel, avec quelques différences notables. Il est un peu frustrant de simplement énumérer ces différences sans avoir la possibilité d'explorer en détail les raisons. Mais il était malgré tout gratifiant de pouvoir mener à bien ce travail de reproduction, si rarement réalisé, malgré son importance reconnue pour la confiance dans les résultats scientifiques.

Tous les scripts et paramètres que nous avons utilisés dans ces expériences seront publiés [8] sous une licence permissive afin que d'autres puissent continuer et étendre notre travail.

Ayant acquis une expérience notable dans la manipulation des modèles de génération de code, nous

8. https ://github.com/cea-list-lasti/CodaBench

nous tournons maintenant vers une participation à la création de jeux de données améliorés et vers l'exploitation des LLM à toutes les étapes de l'ingénierie logicielle.

Remerciements

Ces travaux ont été réalisés grâce au supercalculateur Factory-IA financé par le Conseil Régional d'Île-de-France.

Références

ALMAZROUEI E., ALOBEIDLI H., ALSHAMSI A., CAPPELLI A., COJOCARU R., DEBBAH M., GOFFINET E. *et al.* (2023). The Falcon Series of Open Language Models. arXiv:2311.16867 [cs], DOI : 10.48550/arXiv.2311.16867.

ATHIWARATKUN B., GOUDA S. K., WANG Z., LI X., TIAN Y., TAN M., AHMAD W. U. *et al.* (2023). Multi-lingual Evaluation of Code Generation Models. In *ICLR*.

AUSTIN J., ODENA A., NYE M., BOSMA M., MICHALEWSKI H., DOHAN D., JIANG E. *et al.* (2021). Program Synthesis with Large Language Models. arXiv:2108.07732 [cs], DOI : 10.48550/arXiv.2108.07732.

BELZ A., AGARWAL S., SHIMORINA A. & REITER E. (2021a). A Systematic Review of Reproducibility Research in Natural Language Processing. In P. MERLO, J. TIEDEMANN & R. TSARFATY, Éds., *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, p. 381–393, Online : Association for Computational Linguistics. DOI : 10.18653/v1/2021.eacl-main.29.

BELZ A., POPOVIC M. & MILLE S. (2022). Quantified Reproducibility Assessment of NLP Results. In S. MURESAN, P. NAKOV & A. VILLAVICENCIO, Éds., *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, p. 16–28, Dublin, Ireland : Association for Computational Linguistics. DOI : 10.18653/v1/2022.acl-long.2.

BELZ A., SHIMORINA A., AGARWAL S. & REITER E. (2021b). The ReproGen Shared Task on Reproducibility of Human Evaluations in NLG: Overview and Results. In A. BELZ, A. FAN, E. REITER & Y. SRIPADA, Éds., *Proceedings of the 14th International Conference on Natural Language Generation*, p. 249–258, Aberdeen, Scotland, UK : Association for Computational Linguistics. DOI : 10.18653/v1/2021.inlg-1.24.

BEM D. J. (2011). Feeling the future: experimental evidence for anomalous retroactive influences on cognition and affect. *Journal of personality and social psychology*, **100**(3), 407.

BEN ALLAL L., MUENNIGHOFF N., KUMAR UMAPATHI L., LIPKIN B. & VON WERRA L. (2022). A framework for the evaluation of code generation models. Publication Title : GitHub repository.

BLACK S., BIDERMAN S., HALLAHAN E., ANTHONY Q., GAO L., GOLDING L., HE H. *et al.* (2022). GPT-NeoX-20B: An Open-Source Autoregressive Language Model. arXiv:2204.06745 [cs], DOI : 10.48550/arXiv.2204.06745.

BLACK S., GAO L., WANG P., LEAHY C. & BIDERMAN S. (2021). GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow. DOI : 10.5281/zenodo.5297715.

BRANCO A., CALZOLARI N., VOSSEN P., VAN NOORD G., VAN UYTVANCK D., SILVA J., GOMES L. *et al.* (2020). A Shared Task of a New, Collaborative Type to Foster Reproducibility: A First Exercise in the Area of Language Science and Technology with REPROLANG2020. In N. CALZOLARI, F. BÉCHET, P. BLACHE, K. CHOUKRI, C. CIERI, T. DECLERCK, S. GOGGI, H. ISAHARA, B. MAEGAARD, J. MARIANI, H. MAZO, A. MORENO, J. ODIJK & S. PIPERIDIS, Éds., *Proceedings of the Twelfth Language Resources and Evaluation Conference*, p. 5539–5545, Marseille, France : European Language Resources Association.

CASSANO F., GOUWAR J., NGUYEN D., NGUYEN S., PHIPPS-COSTIN L., PINCKNEY D., YEE M.-H. *et al.* (2022). MultiPL-E: A scalable and extensible approach to benchmarking neural code generation.

CASSANO F., GOUWAR J., NGUYEN D., NGUYEN S., PHIPPS-COSTIN L., PINCKNEY D., YEE M.-H. *et al.* (2023). MultiPL-E: A Scalable and Polyglot Approach to Benchmarking Neural Code Generation. *IEEE Transactions on Software Engineering*, p. 1–17. Conference Name: IEEE Transactions on Software Engineering, DOI : 10.1109/TSE.2023.3267446.

CHEN M., TWOREK J., JUN H., YUAN Q., PINTO H. P. D. O., KAPLAN J., EDWARDS H. *et al.* (2021). Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs], DOI : 10.48550/arXiv.2107.03374.

CHOWDHERY A., NARANG S., DEVLIN J., BOSMA M., MISHRA G., ROBERTS A., BARHAM P. *et al.* (2022). PaLM: Scaling Language Modeling with Pathways. arXiv:2204.02311 [cs], DOI : 10.48550/arXiv.2204.02311.

DEEPSEEK-AI, BI X., CHEN D., CHEN G., CHEN S., DAI D., DENG C. *et al.* (2024). Deepseek LLM: Scaling open-source language models with longtermism. *arXiv:2401.02954*.

DEVLIN J., CHANG M.-W., LEE K. & TOUTANOVA K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, p. 4171–4186, Minneapolis, Minnesota : Association for Computational Linguistics. DOI : 10.18653/v1/N19-1423.

DIRNAGL U. (2019). Rethinking research reproducibility. *The EMBO Journal*, **38**(2), e101117. Publisher : John Wiley & Sons, Ltd, DOI : 10.15252/embj.2018101117.

FENG Z., GUO D., TANG D., DUAN N., FENG X., GONG M., SHOU L. *et al.* (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, p. 1536–1547, Online : Association for Computational Linguistics. DOI : 10.18653/v1/2020.findings-emnlp.139.

FIDLER F. & WILCOX J. (2021). Reproducibility of Scientific Results. In E. N. ZALTA, Éd., *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2021 édition.

FRIED D., AGHAJANYAN A., LIN J., WANG S., WALLACE E., SHI F., ZHONG R. *et al.* (2023). InCoder: A Generative Model for Code Infilling and Synthesis. arXiv:2204.05999 [cs], DOI : 10.48550/arXiv.2204.05999.

GAO L., BIDERMAN S., BLACK S., GOLDING L., HOPPE T., FOSTER C., PHANG J. *et al.* (2020). The Pile: An 800GB Dataset of Diverse Text for Language Modeling. arXiv:2101.00027 [cs], DOI : 10.48550/arXiv.2101.00027.

GITHUB (2023). HumanEval post-processing. https://github.com/bigcode-project/bigcode-evaluation-harness/blob/84b96da31b7f840b55c5733325346176140cdb6b/bigcode_eval/tasks/humaneval.py#L52.

GRATIS VS LIBRE (2024). *Gratis* versus *libre*. https://en.wikipedia.org/w/index.php?title=Gratis_versus_libre&oldid=1217050429#cite_note-1. Page Version ID : 1217050429.

GUO D., REN S., LU S., FENG Z., TANG D., LIU S., ZHOU L. *et al.* (2020). GraphCodeBERT: Pre-training Code Representations with Data Flow. In *International Conference on Learning Representations*.

GUPTA R., PAL S., KANADE A. & SHEVADE S. (2017). DeepFix: fixing common C language errors by deep learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, p. 1345–1351, San Francisco, California, USA : AAAI Press.

HARMAN D. (1993). Overview of the first TREC conference. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '93, p. 36–47, New York, NY, USA : Association for Computing Machinery. DOI : 10.1145/160688.160692.

HENDRYCKS D., BASART S., KADAVATH S., MAZEIKA M., ARORA A., GUO E., BURNS C. *et al.* (2021a). Measuring Coding Challenge Competence With APPS. In J. VANSCHOREN & S. YEUNG, Éds., *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1 : Curran.

HENDRYCKS D., BASART S., KADAVATH S., MAZEIKA M., ARORA A., GUO E., BURNS C. *et al.* (2021b). Measuring coding challenge competence with APPS.

IOANNIDIS J. P. (2005). Why most published research findings are false. *PLoS medicine*, **2**(8), e124.

IYER S., KONSTAS I., CHEUNG A. & ZETTLEMOYER L. (2018). Mapping Language to Code in Programmatic Context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, p. 1643–1652, Brussels, Belgium : Association for Computational Linguistics. DOI : 10.18653/v1/D18-1192.

JIANG A. Q., SABLAYROLLES A., MENSCH A., BAMFORD C., CHAPLOT D. S., CASAS D. D. L., BRESSAND F. *et al.* (2023). Mistral 7B. arXiv:2310.06825 [cs], DOI : 10.48550/arXiv.2310.06825.

JIANG A. Q., SABLAYROLLES A., ROUX A., MENSCH A., SAVARY B., BAMFORD C., CHAPLOT D. S. *et al.* (2024). Mixtral of Experts. arXiv:2401.04088 [cs], DOI : 10.48550/arXiv.2401.04088.

KOCETKOV D., LI R., ALLAL L. B., LI J., MOU C., FERRANDIS C. M., JERNITE Y. *et al.* (2022). The Stack: 3 TB of permissively licensed source code. arXiv:2211.15533 [cs], DOI : 10.48550/arXiv.2211.15533.

LAI Y., LI C., WANG Y., ZHANG T., ZHONG R., ZETTLEMOYER L., YIH S. W.-T. *et al.* (2022). DS-1000: A Natural and Reliable Benchmark for Data Science Code Generation. arXiv:2211.11501 [cs], DOI : 10.48550/arXiv.2211.11501.

LAURENÇON H., SAULNIER L., WANG T., AKIKI C., MORAL A. V. D., SCAO T. L., WERRA L. V. *et al.* (2022). The BigScience ROOTS Corpus: A 1.6TB Composite Multilingual Dataset. In *NeurIPS Datasets and Benchmarks*.

LI R., ALLAL L. B., ZI Y., MUENNIGHOFF N., KOCETKOV D., MOU C., MARONE M. *et al.* (2023). StarCoder: may the source be with you! arXiv:2305.06161 [cs], DOI : 10.48550/arXiv.2305.06161.

LI Y., CHOI D., CHUNG J., KUSHMAN N., SCHRITTWIESER J., LEBLOND R., ECCLES T. *et al.* (2022). Competition-Level Code Generation with AlphaCode. arXiv:2203.07814 Number : arXiv :2203.07814, DOI : 10.48550/arXiv.2203.07814.

LIU J., XIA C. S., WANG Y. & ZHANG L. (2023). Is Your Code Generated by ChatGPT Really Correct ? Rigorous Evaluation of Large Language Models for Code Generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

LOZHKOV A., LI R., ALLAL L. B., CASSANO F., LAMY-POIRIER J., TAZI N., TANG A. *et al.* (2024). StarCoder 2 and The Stack v2 : The Next Generation. arXiv:2402.19173 [cs], DOI : 10.48550/arXiv.2402.19173.

LU S., GUO D., REN S., HUANG J., SVYATKOVSKIY A., BLANCO A., CLEMENT C. *et al.* (2021). CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. In *NeurIPS Datasets and Benchmarks Track*.

NIJKAMP E., HAYASHI H., XIONG C., SAVARESE S. & ZHOU Y. (2023a). CodeGen2: Lessons for Training LLMs on Programming and Natural Languages. arXiv:2305.02309 [cs], DOI : 10.48550/arXiv.2305.02309.

NIJKAMP E., PANG B., HAYASHI H., TU L., WANG H., ZHOU Y., SAVARESE S. *et al.* (2023b). CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. arXiv:2203.13474 Number : arXiv :2203.13474, DOI : 10.48550/arXiv.2203.13474.

OPENAI, ACHIAM J., ADLER S., AGARWAL S., AHMAD L., AKKAYA I., ALEMAN F. L. *et al.* (2023). GPT-4 Technical Report. arXiv:2303.08774 [cs], DOI : 10.48550/arXiv.2303.08774.

PENEDO G., MALARTIC Q., HESSLOW D., COJOCARU R., CAPPELLI A., ALOBEIDLI H., PANNIER B. *et al.* (2023). The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*. arXiv: 2306.01116.

PENG Q., CHAI Y. & LI X. (2024). HumanEval-XL: A multilingual code generation benchmark for cross-lingual natural language generalization. In N. CALZOLARI, M.-Y. KAN, V. HOSTE, A. LENCI, S. SAKTI & N. XUE, Éds., *2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, p. 8383–8394, Torino, Italia : ELRA and ICCL.

REN S., GUO D., LU S., ZHOU L., LIU S., TANG D., SUNDARESAN N. *et al.* (2020). Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.

ROZIÈRE B., GEHRING J., GLOECKLE F., SOOTLA S., GAT I., TAN X. E., ADI Y. *et al.* (2023). Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs], DOI : 10.48550/arXiv.2308.12950.

SCAO T. L., FAN A., AKIKI C., PAVLICK E., ILIĆ S., HESSLOW D., CASTAGNÉ R. *et al.* (2023). BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. arXiv:2211.05100 [cs], DOI : 10.48550/arXiv.2211.05100.

WANG B. & KOMATSUZAKI A. (2021). GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

WANG S., LI Z., QIAN H., YANG C., WANG Z., SHANG M., KUMAR V. *et al.* (2023). ReCode: Robustness Evaluation of Code Generation Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, p. 13818–13843, Toronto, Canada : Association for Computational Linguistics.

WANG Y., WANG W., JOTY S. & HOI S. C. (2021). CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, p. 8696–8708, Online and Punta Cana, Dominican Republic : Association for Computational Linguistics. DOI : 10.18653/v1/2021.emnlp-main.685.

Xu F. F., Alon U., Neubig G. & Hellendoorn V. J. (2022). A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, MAPS 2022, p. 1–10, New York, NY, USA : Association for Computing Machinery. DOI : 10.1145/3520312.3534862.

Xu Y., Su H., Xing C., Mi B., Liu Q., Shi W., Hui B. *et al.* (2023). Lemur: Harmonizing Natural Language and Code for Language Agents. arXiv:2310.06830 [cs], DOI : 10.48550/arXiv.2310.06830.

Zdar & Allal L. B. (2023). Instruct humaneval.

Zheng Q., Xia X., Zou X., Dong Y., Wang S., Xue Y., Wang Z. *et al.* (2023). CodeGeeX: A Pre-Trained Model for Code Generation with Multilingual Evaluations on HumanEval-X. https ://keg.cs.tsinghua.edu.cn/codegeex/, DOI : 10.48550/arXiv.2303.17568.

Zheng Z., Ning K., Wang Y., Zhang J., Zheng D., Ye M. & Chen J. (2024). A Survey of Large Language Models for Code: Evolution, Benchmarking, and Future Trends. arXiv:2311.10372 [cs], DOI : 10.48550/arXiv.2311.10372.

Zohny H., McMillan J. & King M. (2023). Ethics of generative AI. *Journal of Medical Ethics*, **49**(2), 79–80. Publisher : Institute of Medical Ethics Section : Editorial, DOI : 10.1136/jme-2023-108909.

Annexes

A Modèles

cf. tableau 9.

B Benchmarks

Le tableau 10 donne une vue d'ensemble des benchmarks existant pour l'évaluation de la génération de code. HumanEval est à notre connaissance le premier benchmark évaluant les programmes générés par leur capacité à être exécutés et à passer des tests unitaires. Il contient 164 descriptions de tâches à réaliser ainsi que l'entête de la fonction python à générer. Chaque tâche est accompagnée de plusieurs tests unitaires qui doivent passer pour que le programme soit accepté. Les benchmarks Multilingual HumanEval, InstructHumanEval, HumanEval-X, MXBP et EvalPlus sont des variations de HumanEval : des traductions vers d'autres langages, une adaptation pour évaluer particulièrement les modèles d'instruction, etc. MTPB (Multi-Turn Programming Benchmark) (Nijkamp *et al.*, 2023b) contient 115 problèmes écrits par des experts, chacun comprenant une description multi-étapes en langage naturel (invite). Pour résoudre un problème, un modèle doit synthétiser des sous-programmes fonctionnellement corrects (1) suivant la description à l'étape actuelle et (2) en tenant compte des descriptions et des sous-programmes synthétisés lors des étapes précédentes. MBPP (Mostly Basic Programming Problems) (Austin *et al.*, 2021) est composé de 974 courts programmes Python construits par crowd-sourcing auprès de travailleurs ayant des connaissances de base en Python. Il a été demandé aux participants d'écrire un court énoncé de problème, une seule fonction Python autonome répondant au problème spécifié, et trois cas de test vérifiant la correction sémantique de la fonction. Le jeu de données APPS (Hendrycks *et al.*, 2021b) se compose de problèmes collectés à partir de différents sites Web de codage en accès libre tels que CodeForce, Kattis…Il tente de refléter la façon dont les programmeurs humains sont évalués en posant des problèmes de codage en langage naturel non restreint. Le jeu de données se compose de 10 000 problèmes de codage au total, avec 131 836 cas de test pour vérifier les solutions et 232 444 solutions de référence écrites par des humains. La longueur moyenne d'un problème est de 293,2 mots. Le jeu de données Multipl-E (Cassano *et al.*, 2022) consiste en une traduction automatique des jeux de données HumanEval et MBPP vers d'autres langues. Il supporte 18 langages de programmation. ReCode, qui explore la robustesse de la génération de code avec des perturbations dans l'invite (Wang *et al.*, 2023), n'est pas un banc d'essai en soi. C'est une méthode pour tester la robustesse des modèles sur d'autres benchmarks, à savoir HumanEval et MBPP. Elle applique des transformations aux invites pour les rendre réalistement plus difficiles.

C Résultats sur HumanEval

cf. tableau 11.

D Résultats sur Multipl-E

cf. tableau 12.

	Propriétés des Modèles			Jeux de Données			Évaluation	
	Open	Multi langages	#Params	Source	Langages	Tailles	Éval. Multi-lang.	Benchmarks
Codex (Chen *et al.*, 2021)	N	N	12B	Collected	Python	Code : 159GB	N	HumanEval, APPS
AlphaCode (Li *et al.*, 2022)	N	Y	41B	Collected	12 langs	Code : 715.1GB	N	HumanEval, APPS, CodeContest
PaLM-Coder (Chowdhery *et al.*, 2022)	N	Y	8B, 62B, 540B	Collected	Multiple	Text : 741B tokens Code : 39GB (780B tokens trained)	Y	HumanEval, MBPP, DeepFix
PolyCoder (Xu *et al.*, 2022)	Y	Y	2.7B	Collected	12 langs	Code : 253.6GB	N	HumanEval
GPT-Neo (Black *et al.*, 2021)	Y	Y	1.3B, 2.7B	The Pile	Multiple	Text : 730GB Code : 96GB (400B tokens trained)	N	HumanEval
GPT-NeoX (Black *et al.*, 2022)	Y	Y	20B	The Pile	Multiple	Text : 730GB Code : 96GB (473B tokens trained)	N	HumanEval
GPT-J (Wang & Komatsuzaki, 2021)	Y	Y	6B	The Pile	Multiple	Text : 730GB Code : 96GB (473B tokens trained)	N	HumanEval
Incoder (Fried *et al.*, 2023)	I	Y	1.3B, 6.7B	Collected	ROOTS28 langs	Code : 159GB StackOverflow : 57GB (60B tokens trained)	N	HumanEval, MBPP CodeXGLUE
CodeGen-Multi (Nijkamp *et al.*, 2023b)	I	Y	6.1B, 16.1B	BigQuery	6 langs	Code : 150B tokens Text : 355B tokens (1000B tokens trained)	N	HumanEval, MTPB
CodeGen-Mono (Nijkamp *et al.*, 2023b)	I	N	6.1B, 16.1B	BigPython	Python	Code : 150B tokens Text : 355B tokens (1300B tokens trained)	N	HumanEval, MTPB
CodeGeeX (Zheng *et al.*, 2023)	IRO	Y	13B	The Pile CodeParrot Collected	23 langs	Code : 158B tokens (850B tokens trained)	Y	HumanEval-X, HumanEval MBPP, CodeXGLUE, XLCoST
CodeGeeX2 (Zheng *et al.*, 2023)	IRO	Y	6B	The Pile, CodeParrot, Collected	>100	CodeGeeX+600B	Y	HumanEval, HumanEval-X DS1000
StarCoder (Li *et al.*, 2023)	Y	Y	15.5B	The Stack v1.2	>80	1Trillion	Y	HumanEval-X, HumanEval MBPP, MultiPL-E
Bloom (Scao *et al.*, 2023)	Y	Y	176B	ROOTS	13	350B	N	HumanEval
Falcon (Almazrouei *et al.*, 2023)	NC	Y	180B	RefinedWeb	?	Text : 2,700BT, Code : 115BT	N	HumanEval
CodeGen2 (Nijkamp *et al.*, 2023a)	I	Y	7B	StarCoderData	86	Code : 250BT		HumanEval, HumanEval-Infill, XSum, CodeXGLUE, SuperGLUE
Llama2 (Rozière *et al.*, 2023)	NF	Y	34B	"publicly av. code"	>= 7	500BT (from Llama2)	Y	HumanEval, MBPP, APPS, MultiPL-E
Code Llama (Rozière *et al.*, 2023)	NF	Y	34B	"publicly av. code"	>= 7	500BT (from Llama2)	Y	HumanEval, MBPP, APPS, MultiPL-E
Mistral7B (Jiang *et al.*, 2023)	I	Y	7B	U	U	U	N	HumanEval, MBPP
Mistral8x7B (Jiang *et al.*, 2024)	I	Y	12B	U	U	U	N	HumanEval, MBPP
Lemur-70B (Xu *et al.*, 2023)	NF	Y	70B	The Stack...	Scripts	Llama2+90BT	Y	HumanEval, MBPP, Spider, MultiPL-E, DS-1000
DeepSeek (DeepSeek-AI *et al.*, 2024)	IR	Y	1-33B	StarCoderData method	89	2T tokens	Y	HumanEval, MBPP, DS-1000
ChatGPT-4 (OpenAI *et al.*, 2023)	N	Y	U	U	U	U	Y	HumanEval

TABLE 9 – LLM préentraînés de l'état de l'art relatifs à des langages de programmation. Données initiales issues de (Zheng *et al.*, 2023)

Ouverture : IR=inférence seulement avec restrictions ; IRS=inférence recherche seulement (tout le reste sous licence commerciale et source fermée ; NC=Non commercial ; I=inférence ouverte/libre, pas d'information sur l'entraînement ; NL=Ouvert mais non libre. Tous : NP=Non publié/Inconnu

	Langages	Caractéristiques	Nb problèmes
HumanEval (Chen *et al.*, 2021)	Python	NL prompt+a set of unit tests	164
HumanEvalX (Zheng *et al.*, 2023)	5	Same as HE, translated to other languages	164×5
APPS (Hendrycks *et al.*, 2021a)	Python	NL prompt+a set of unit tests +solutions	10k problems,132k test cases, 232k solutions
MBPP Mostly Basic Programming Problems (Austin *et al.*, 2021)	Python	programming problems with test cases and example solutions	974
MTPB Multi-Turn Programming Benchmark (Nijkamp *et al.*, 2023b)	Python	Programming problem expressed as several steps with unit tests at each step	115
ReCode (Wang *et al.*, 2023)	Python	Reformulations of HumanEval and MBPP	164 + 974
DS-1000 (Lai *et al.*, 2022)	Python	Data science problems from SO, test cases+constraints	1000
MultiPL-E (Cassano *et al.*, 2023)	19	translation of HumanEval and MBPP to other languages by compilers	164 + 974
CodeXGLUE (Lu *et al.*, 2021)	Python, Java	Concatenation of other datasets, code search for Python, Code Gen for Java	
DeepFix (Gupta *et al.*, 2017)	C	Train=correct program +artificialy mutated wrong. Test=student wrong programs	
BigCode Evaluation Harness (Ben Allal *et al.*, 2022)	Several	A framework allowing to run all above benchmarks and others on HuggingFace models	A lot
EvalPlus (Liu *et al.*, 2023)	Python	Generates new test cases for HumanEval problems ($\times 80$)	164
InstructHumanEval (Zdar & Allal, 2023)	Python	Reformulates HumanEval as prompts adapted for instructed LLMs	164
MultiLingual HumanEval (Athiwaratkun *et al.*, 2023)	13	Transpilation of HumanEval	164
MXBP (Athiwaratkun *et al.*, 2023)	13	Transpilation of MBPP	974

TABLE 10 – Jeux de données pour la génération de code. SO=StackOverflow.

Modèle	Rapporté	Reproduit
CodeLlama-34B	48,8	47,0
Phind-CodeLlama-34B-v2	73,6	72,7
GPT-J	11,6	11,0
CodeGen-16B-mono	29,3	31,1
CodeGen2-7B-multi	18,8	19,6
CodeGeeX	22,9	22,8
DeepSeek-Base-1.3B	34,8	33,5
DeepSeek-Instruct-33B	79,3	61,3
DeepSeek-Instruct-6.7B	–	71,3
Llama2-13B	18,3	6,7
Llama2-70B	29,9	20,6
Mistral 7B	30,5	6,3
Mixtral 8x7B	40,2	–
Mixtral 8x7B-Instruct	–	49,5
StarCoder-15B	33,6	33,8
StarCoder2-3B	31,7	30,8
StarCoder2-7B	35,4	34,6
StarCoder2-15B	46,3	44,7
StarCoder2-3B fine-tuned the-stack-smol*	–	30,7
StarCoder2-3B self-instruct*	–	32,7
Dolphin :starcoder2-15B	–	54,5
StarCoder2-3B Magicoder-OSS-Instruct-75K *	–	34,4

TABLE 11 – Résultats pour HumanEval. * indique les modèles que nous avons affinés ou instruits.

Langage	DeepSeek-Instruct-33B		StarCoder-15B		StarCoder2-7B	
	Rapporté	Reproduit	Rapporté	Reproduit	Rapporté	Reproduit
Python	79,3	–	33,6	33,7	–	32,9
C++	68,9	62,6	31,6	30,4	33,6	40,2
C#	74,1	40,0	21,0	20,8	20,7	24,3
D	–	–	13,6	13,0	15,1	16,5
Go	–	0,0	17,6	0,0	20,2	0,6
Java	73,4	51,3	30,2	28,2	29,4	35,5
Julia	–	–	23,0	–	20,4	20,0
JavaScript	73,9	62,4	30,8	30,9	35,4	35,9
Lua	–	48,3	23,9	24,0	30,7	30,8
PHP	72,7	52,3	26,1	25,5	30,6	32,0
Perl	–	38,3	17,3	16,8	16,6	17,0
R	–	39,4	15,5	16,1	16,7	21,4
Ruby	–	45,1	1,2	1,6	28,3	28,6
Racket	–	26,1	0,1	10,0	11,6	12,9
Rust	–	49,1	21,8	22,1	29,6	29,2
Scala	–		27,6	25,9	19,5	18,5
Bash	43,0	–	10,5	11,4	12,2	14,9
Swift	–	–	22,7	0,0	26,1	0,4
TypeScript	67,9	–	32,3	–	36,3	35,1

TABLE 12 – Résultats pour MultiPL-E