Evaluating LLMs Efficiency Using Successive Attempts on Binary-Outcome Tasks

Mohamed Amine El Yagouby[1,2] Abdelkader Lahmadi[1] Mehdi Zakroum[2]
Olivier Festor[1] Mounir Ghogho[2]
(1) Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
(2) Université Internationale de Rabat, TICLab, 11103 Rabat, Morocco
{mohamed-amine.el-yagouby, abdelkader.lahmadi, olivier.festor}@loria.fr
mehdi.zakroum@uir.ac.ma, mounir.ghogho@um6p.ma

ABSTRACT

Evaluating Large Language Models (LLMs) using single-attempt metrics like Success Rate (SR) overlooks their capacity for iterative problem solving. In tasks with binary outcomes (success or failure), such as coding or planning, LLMs often benefit from multiple attempts. Existing multi-attempt metrics like pass@k and success@k account for eventual success but ignore how efficiently it is achieved, making them more costly. We propose a new evaluation method with Successive Multiple Attempts, where a maximum number of retries is fixed, and introduce our **Success Efficiency (SE)** metric, which captures both success and efficiency in a single value by rewarding earlier successes and penalizing delays. Tested using the HumanEval dataset across six LLMs, SE captures how quickly an LLM solves tasks, which existing metrics do not offer. This work complements existing evaluation methods by measuring not only whether LLMs succeed but also how efficiently they do so.

RÉSUMÉ

Évaluation de l'efficacité des LLMs à l'aide de tentatives successives sur des tâches à résultat binaire.

L'évaluation des grands modèles de langage (LLMs) à l'aide de métriques à tentative unique comme le taux de succès (SR) ne tient pas compte de leur capacité à résoudre les problèmes de manière itérative. Dans les tâches aux résultats binaires (succès ou échec), telles que le codage ou la planification, les LLM bénéficient généralement de tentatives multiples. Les mesures multi-tentatives existantes, telles que pass@k et success@k, tiennent compte du succès éventuel, mais ignorent l'efficacité avec laquelle il est obtenu, ce qui les rend plus coûteuses. Nous proposons une nouvelle méthode d'évaluation avec des tentatives multiples successives, où un nombre maximum de tentatives est fixé, et introduisons la métrique **Efficacité du succès (SE)**, qui capture à la fois le succès et l'efficacité en une seule valeur en récompensant les succès antérieurs et en pénalisant les retards. Testée sur le jeu de données HumanEval avec six LLMs, SE capture la rapidité avec laquelle un LLM résout les tâches, ce que les métriques existantes n'offrent pas. Ce travail complète les méthodes d'évaluation existantes en mesurant non seulement la réussite des LLMs, mais aussi l'efficacité avec laquelle ils y parviennent.

KEYWORDS: LLM Evaluation, Success Efficiency.

MOTS-CLÉS : Évaluation du LLM, Efficacité du succès.

ARTICLE : **Accepté à** EvalLLM 2025.

1 Introduction

Understanding the practical utility of Large Language Models (LLMs) requires evaluating them across a wide range of tasks. Many of these tasks, including code generation (Hendrycks *et al.*, 2021), theorem proving (Polu & Sutskever, 2020), logical reasoning (Yu *et al.*, 2020), and instruction following (Ouyang *et al.*, 2022), can be framed as **binary outcome tasks**, where each evaluation result of an LLM is either a success or a failure. Traditionally, such tasks have been evaluated using a **single-shot** paradigm, in which an LLM is given a single attempt to solve a task. This yields simple metrics like the **Success Rate (SR)** described in Section 2, which captures the rate at which the LLM succeeds on the first Attempt across a given list of tasks.

While SR and other similar metrics are informative, they overlook an important dimension of LLMs behavior : the capacity for **multi-attempt problem solving** (Li *et al.*, 2022). In practical deployments, such as interactive assistants (Foosherian *et al.*, 2023), automated coding tools (Li *et al.*, 2022), or planning agents (Yao *et al.*, 2023), LLMs are often allowed multiple chances to revise or regenerate responses. To reflect this, recent studies have introduced evaluation frameworks that support multiple attempts, either through independent sampling, such as **pass@k** (Chen *et al.*, 2021) or sequential retries, such as **success@k** (Kulal *et al.*, 2019). These metrics assess whether an LLM can solve a task within a fixed number of attempts, but treat all successful completions equally, regardless of whether the solution occurred early or late in the sequence. Consequently, they fail to capture the *efficiency* of problem solving (as defined in Section 2), which is a critical factor in latency-sensitive and resource-constrained applications.

To address this limitation, we propose an evaluation methodology based on **Successive Multiple Attempts**, in which an LLM is allowed to iteratively attempt a task up to a fixed maximum number of attempts (MaxA). This controlled retry setting enables us to measure not only whether an LLM can eventually solve a task, but also *how rapidly* and it does so.

At the core of our methodology lies the **Success Efficiency (SE)** detailed in Section 2, a novel metric that unifies both success and efficiency into a single, interpretable measure. Unlike metrics such as pass@k or success@k, SE provides a continuous score that penalizes delayed success. A detailed comparison between our proposed **SE** metric and these related metrics is given in Section 3.

We evaluated six state-of-the-art (SotA) LLMs using our methodology on the HumanEval dataset (Chen *et al.*, 2021), a widely used benchmark for code generation. The evaluation setup and results are detailed in Section 4, where we compared SE to the traditional SR metric. The results show that SE provides a more informative assessment by distinguishing between LLMs that rapidly succeed and those that required multiple retries. This underscores SE's value as a metric for evaluating LLMs in iterative, retry-enabled scenarios.

2 Evaluation Methodology

To rigorously evaluate the performance of LLMs on a fixed set of tasks, we introduce three complementary metrics. These metrics assess both the effectiveness (how often the LLM succeeds) and the efficiency (how quickly it succeeds within a limited number of attempts).
Let MaxA ≥ 1 denote the maximum number of allowed attempts per task. The evaluation metrics are defined as follows :

1. **Success Rate (SR)**

$$\text{SR} = \frac{\text{Successful Tasks}}{\text{Total Tasks}}, \quad \text{where } 0 \leq \text{SR} \leq 1$$

SR represents the proportion of tasks that the LLM eventually solves, regardless of how many attempts are needed (within the allowed limit). A higher SR reflects better task-solving capability.

2. **Average Task Completion Attempts (AvgTCA)**

$$\text{AvgTCA} = \frac{1}{\text{Successful Tasks}} \sum \text{TCA}, \quad \text{where } 1 \leq \text{AvgTCA} \leq \text{MaxA}$$

TCA (Task Completion Attempts) reflects how many attempts the LLM needed to complete a given task. AvgTCA measures how many attempts, on average, the LLM needs to successfully complete the tasks it eventually solves. Lower AvgTCA values indicate faster and more efficient task sloving.

3. **Success Efficiency (SE)**

$$\text{SE} = \frac{\text{SR}}{\text{AvgTCA}^{\left(\frac{\text{AvgTCA}-1}{\text{MaxA}-1}\right)}}, \quad \text{where } 0 \leq \text{SE} \leq \text{SR}$$

The SE score combines both success and efficiency into a single value. For any $SR > 0$, SE reaches its *maximum* (equal to SR) when AvgTCA $= 1$, meaning all successful tasks were completed on the first attempt. In contrast, it reaches its *minimum* when AvgTCA $=$ MaxA, reflecting highly inefficient completions.

The exponent in the formula of SE, $\left(\frac{\text{AvgTCA}-1}{\text{MaxA}-1}\right)$, measures how far the LLM's AvgTCA deviates from the ideal value of 1 (perfect efficiency) normalized by the maximum possible deviation $(\text{MaxA}-1)$, Within the SR calculation, this exponent acts as a normalization penalty in $[0, 1]$, applying minimal penalization when (AvgTCA $= 1$), and gradually increasing as AvgTCA approaches the allowed maximum (MaxA). This formulation ensures that SE remains :

— **Bounded** : It cannot exceed SR or fall below zero, preserving numerical stability.
— **Interpretable** : Maintains a direct relationship with SR while incorporating AvgTCA to measure efficiency, which enables meaningful comparisons.
— **Adaptable** : It scales consistently across different values of MaxA, allowing fair evaluation under varying task constraints.

As a result, SE rewards LLMs that are not only accurate but also solve tasks with minimal effort.

3 Existing Evaluation Metrics Review

Existing evaluation metrics for binary-outcome tasks, such as SR, typically measure whether a task is eventually solved. Some metrics, like Pass@k and Success@k, account for the number of attempts, but they do not capture how efficiently a solution is reached in terms of successive attempts. In Table 1, we compared these metrics based on their ability to capture success, attempts, and efficiency :

Metric	Success	Attempts	Efficiency
SR	✓		
Pass@k	✓	✓(random)	
Success@k	✓	✓(sequential)	
SE (Ours)	✓	✓(sequential)	✓

TABLE 1 – Comparison of evaluation metrics for binary-outcome tasks in capturing success, attempts and efficiency

SR, described in Section 2, measures the fraction of tasks eventually completed by an LLM, regardless of how many tries it takes. It is simple and interpretable but ignores any notion of Attempts or how long success takes.
Pass@k (Chen *et al.*, 2021), commonly used in code generation, checks if at least one of k independently sampled outputs solves the task. It accounts for success across multiple random attempts but ignores the order or efficiency of these attempts.
Success@k (Kulal *et al.*, 2019), improves on this by considering sequential attempts, such as iterative refinements or retries. It measures the percentage of tasks completed within the first k ordered attempts. While this introduces some temporal structure, it still gives equal credit to solving a task on the first or the k^{th} attempt and does not penalize inefficiency.
Our proposed **SE** metric addresses this gap by integrating both *success* and *efficiency* into a single metric. As detailed in Section 2, it rewards LLMs that solve tasks quickly and penalizes those that require more attempts, making it especially suited for evaluating LLMs in practical applications that enables Attempting.

4 LLMs Evaluation Using Our Metrics on HumanEval

4.1 Setup : Dataset, LLMs, Prompt Template, and Configurations

To evaluate LLMs on our metrics introduced in Section 2, we used the first 100 tasks from the HumanEval dataset (Chen *et al.*, 2021). This data set contains a diverse set of Python programming tasks and is adopted as a standard benchmark for evaluating the quality of code generation. Each task consists of a Python function signature, a natural language docstring, and a corresponding unit test suite. A solution is considered successful if it executes without errors and passes all unit tests. The binary nature of this dataset (where a generated solution either passes all test cases or fails), makes it particularly well-suited for evaluating our approach, which is designed for binary-outcome tasks.

We selected six state-of-the-art LLMs (SotA) through the OpenRouter API [1], including a commercial LLM (OpenAI's GPT-4) and five open source LLMs, as listed in Table 2. Each LLM was prompted using a successive-attempt strategy. In this setup, if the initial response fails, the LLM is re-prompted using the feedback template in Figure 1, which includes the failure message along with the history of previous attempts. This simulates a correction loop in which the LLM can refine its solution based on prior errors. Each LLM was allowed a maximum of MaxA $= 2$ attempts per task, ensuring a fair comparison under constrained retry conditions.

1. https://openrouter.ai

```
You are an excellent programmer, solving this
coding task : {task_description}
You previously attempted to solve this task
without success.
Below are the previous attempts:
Attempt 1: {responses_history[0]}
Attempt 2: {responses_history[1]}
...
These attempts failed. Provide the correct
solution for the task above. Return only the
code.
```

FIGURE 1 – Feedback template to prompt the LLM to correct its previous attempts.

To introduce controlled variability between attempts, We set the temperature parameter to 0.1 for all evaluated LLMs. This low value encourages slight randomness in generation while maintaining consistency and high-quality outputs. The choice is especially important given that some HumanEval tasks may have been present in the training data of these recent LLMs, as this dataset has been publicly available since 2021. Using a low temperature helps mitigate deterministic overfitting in such cases.

All code, along with the full experimental configuration, data and LLMs prompt templates for reproducibility, is available in our GitHub repository. [2]

4.2 Results

The LLMs evaluation results on the first 100 tasks of the HumanEval dataset under successive attempts, summarized in Table 2, shows that while SR captures whether a task is eventually solved, SE offers a more nuanced view by penalizing inefficiency in reaching a solution.

LLM	**SR**	**AvgTCA**	**SE**
mistralai/codestral-2501	0.95	1.074	0.945
meta-llama/llama-3.3-70b-instruct	0.94	1.149	0.921
openai/gpt-4	0.84	1.274	0.786
google/palm-2-codechat-bison	0.76	1.092	0.754
meta-llama/llama-3-8b-instruct	0.79	1.342	0.714
mistralai/mistral-7b-instruct	0.62	1.355	0.557

TABLE 2 – Success Rate (SR), Average Task Completion Attempts (AvgTCA), and Success Efficiency (SE) of six SotA LLMs on the first 100 tasks from HumanEval.

For instance, although meta-llama/llama-3-8b-instruct achieves a slightly higher SR of 0.79 compared to google/palm-2-codechat-bison's 0.76, its SE is lower (0.714 vs. 0.754) due to requiring more attempts on average. This indicates that SR alone may overestimate the practical effectiveness

2. https://github.com/amine-elyagouby/LLM-Efficiency-Eval

of a LLM when attempts are costly. A similar pattern emerges when comparing mistralai/codestral-2501 and meta-llama/llama-3.3-70b-instruct. While both LLMs achieve nearly identical SRs (0.95 and 0.94 respectively), codestral-2501 has a higher SE (0.945 vs. 0.921), reflecting its greater efficiency in task completion. Remarkably, this open-source LLM outperforms even proprietary LLMs like openai/gpt-4, potentially due to its training optimization on code-related tasks.

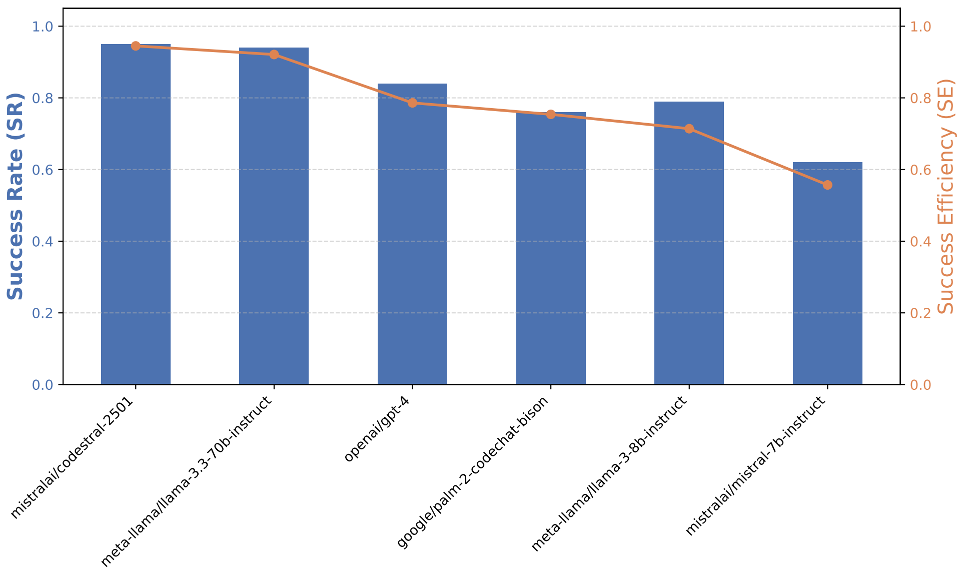

FIGURE 2 – Success Rate (SR) vs. Success Efficiency (SE) for LLMs on our dataset (first 100 HumanEval tasks), with SE highlighting efficiency differences beyond SR.

Furthermore, even commercial high-performing LLMs like openai/gpt-4 has efficiency trade-offs. Despite its good SR of 0.84, its relatively high average attempts (1.274) lead to an SE of just 0.786. This suggests that GPT-4's robustness may come at the cost of increased iteration, which could be a limitation in latency-sensitive applications.

Visualizing both SR and SE in Figure 2 further illustrates these cases where LLMs with similar SRs diverge in efficiency. Notably, LLMs such as mistralai/codestral-2501 not only succeed frequently but also do so with minimal retries, making them well-suited for deployment in code generation scenarios. In contrast, LLMs with high SR but lower SE may be less practical, particularly in contexts where response time is critical.

5 Conclusion and Future Work

In this paper, we propose a novel evaluation metric, Success Efficiency (SE), to assess the performance of LLMs in multi-attempt settings, capturing both task completion and efficiency. This makes SE particularly suitable for practical applications where responsiveness and resource usage are critical. Experiments on the HumanEval dataset demonstrate SE's ability to highlight performance differences which are overlooked by existing metrics. Future work may includes extending SE to non-binary tasks like open-ended generation or multi-step reasoning, developing adaptive retry strategies based on task complexity, and integrating cost-awareness such as inference time or compute usage, to better promote sustainable LLM deployment.

Acknowledgment

This work is part of a collaborative research project between *Université de Lorraine* (France) and *Université Internationale de Rabat* (Morocco), supported by *Lorraine Université d'Excellence* under the France 2030 Program, reference ANR-15-IDEX-04-LUE.

References

CHEN M., TWOREK J., JUN H., YUAN Q., PINTO H. P. D. O., KAPLAN J., EDWARDS H., BURDA Y., JOSEPH N., BROCKMAN G. *et al.* (2021). Evaluating large language models trained on code. *arXiv preprint arXiv :2107.03374*.

FOOSHERIAN M., PURWINS H., RATHNAYAKE P., ALAM T., TEIMAO R. & THOBEN K.-D. (2023). Enhancing pipeline-based conversational agents with large language models. *arXiv preprint arXiv :2309.03748*.

HENDRYCKS D., BASART S., KADAVATH S., MAZEIKA M., ARORA A., GUO E., BURNS C., PURANIK S., HE H., SONG D. & STEINHARDT J. (2021). Measuring Coding Challenge Competence with APPS. In *NeurIPS Datasets and Benchmarks Track*. arXiv :2105.09938.

KULAL S., PASUPAT P., CHANDRA K., LEE M., PADON O., AIKEN A. & LIANG P. (2019). SPoC : Search-Based Pseudocode to Code. In *Advances in Neural Information Processing Systems (NeurIPS) 32*.

LI Y., CHOI D., CHUNG J., KUSHMAN N., SCHRITTWIESER J., LEBLOND R., ECCLES T., KEELING J., GIMENO F. *et al.* (2022). Competition-Level Code Generation with AlphaCode. *Science*, **378**(6624), 1092–1097. DOI : 10.1126/science.abq1158.

OUYANG L., WU J., JIANG X., ALMEIDA D., WAINWRIGHT C. L., MISHKIN P., ZHANG C. *et al.* (2022). Training Language Models to Follow Instructions with Human Feedback. In *Advances in Neural Information Processing Systems (NeurIPS) 35*.

POLU S. & SUTSKEVER I. (2020). Generative Language Modeling for Automated Theorem Proving. *arXiv preprint arXiv :2009.03393*.

YAO S., ZHAO J., YU D., DU N., SHAFRAN I., NARASIMHAN K. & CAO Y. (2023). React : Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

YU W., JIANG Z., DONG Y. & FENG J. (2020). ReClor : A Reading Comprehension Dataset Requiring Logical Reasoning. In *8th International Conference on Learning Representations (ICLR)*.