Evaluating LLM Generated Task Codes

Sanjana Yasna
Computer Science Department
Smith College
Northampton, USA
0009-0007-2899-9660

Simon Garcia de Gonzalo
Sandia National Laboratory
Albuquerque, USA
0000-0002-5699-1793

Michael Robson
Computer Science Department
Smith College
Northampton, USA
0000-0002-4859-0033

***Abstract*—Asynchronous many-task (AMT) programming models offer significant advantages over traditional approaches like MPI for irregular and dynamic workloads, yet remain difficult to adopt due to limited documentation and code examples. We extend the ParEval benchmark to evaluate the ability of state-of-the-art large language models to generate code for HPX, a representative AMT runtime, testing both closed-source models (GPT-5, Claude Sonnet 4.5) and leading open-source code models on existing parallel programming tasks plus new AMT-specific prompts. Our evaluation reveals a pronounced performance gap: Claude Sonnet 4.5 achieves the highest strict_pass@1 of 0.61 followed by GPT-5 with 0.53, while open-source models lag behind on the overall benchmark and frequently resort to conventional multithreaded C++ solutions rather than HPX abstractions. This work establishes a comprehensive baseline for LLM performance on under-resourced AMT languages and identifies areas for improvement in developing intelligent assistants for advanced parallel programming frameworks.**

***Index Terms*—large language models, parallel programming, benchmark testing, high performance computing**

I. Introduction

As computations scale and exhibit dynamic workloads and irregular execution patterns, evenly distributed and fixed workloads managed through interfaces like MPI do not scale well. Asynchronous many-task (AMT) models offer flexible alternatives in which applications are executed via task dependencies with more fine-grained communication [1]. Despite these advantages, AMTs have limited documentation and scarce code examples [2], making them low-resource languages (i.e., limited training data) that are difficult for developers to adopt.

Our goal is to evaluate the ability of large language models (LLMs) to generate single-function code for parallel asynchronous many-task languages such as Charm++ [3], HPX [4], and Legion [5]. This work lays the foundation for future efforts to benchmark LLM performance on other AMT languages and evaluate the efficacy of code generated in these under-resourced languages.

Current parallel code evaluation frameworks, such as ParEval [6], primarily examine the ability of LLMs to generate code in MPI, OpenMP, and other well-established parallel programming models. We extend the ParEval framework by adding a new class of prompts tailored to task-based libraries and design our benchmark around HPX [4], a representative AMT runtime. We evaluate the ability of current LLMs to generate HPX code for both existing and newly added prompts, measuring both scalability and correctness. ChatHPC [7] measures the ability of a fine-tuned Code Llama LLM to translate OpenMP, CUDA, and HIP code to IRIS. While they take a similar approach, they do not measure generated code runtimes and don't make use of the ParEval suite.

The contributions of this work include: a study of available corpora for training under-resourced AMT languages, the addition of HPX evaluation to existing ParEval prompts, an extension of the ParEval benchmark with two new task-specific prompt categories (locking/contention and futures/promises) and an updated evaluation of GPT-5, Claude Sonnet 4.5, and previous best open source models performance on our extended ParEval HPX benchmark.

II. Methods

A. Identifying AMT

Below, in Table I, we have compiled code availability statistics for a variety of open source AMTs on GitHub. We query directly from GitHub the number of program-level (pertaining to individual programming modules) and file-level (pertaining to individual file code samples aside form headers) for individual AMT languages. HPX has the most program-level and file-level code among the AMTs we queried. We have also included both Kokkos [8] and Raja [9] as popular parallel programming frameworks that are not AMTs to compare general levels of code availability.

AMT	Programs	Files
HPX [4]	11.1k	23.6k
CHARM++ [3]	920	2.3k
Legion [5]	54	2.5k
StarPU [10]	120	1.8k
Kokkos [8]	6.8k	41.9k
Raja [9]	1.3k	5.6k

TABLE I: AMTs, numbers of programs and number of files as reported by direct GitHub search.

B. Model Selection

In addition to selecting a target AMT to extend ParEval, we selected a variety of LLMs to evaluate their ability to generate AMT (HPX) code. Our selection was based on several factors: prior performance in published benchmarks (particularly those previously evaluated by ParEval), open-source availability, and coding ability as measured by LMArena.ai [11]. For closed models we selected OpenAI's ChatGPT-5 and Claude's Sonnet

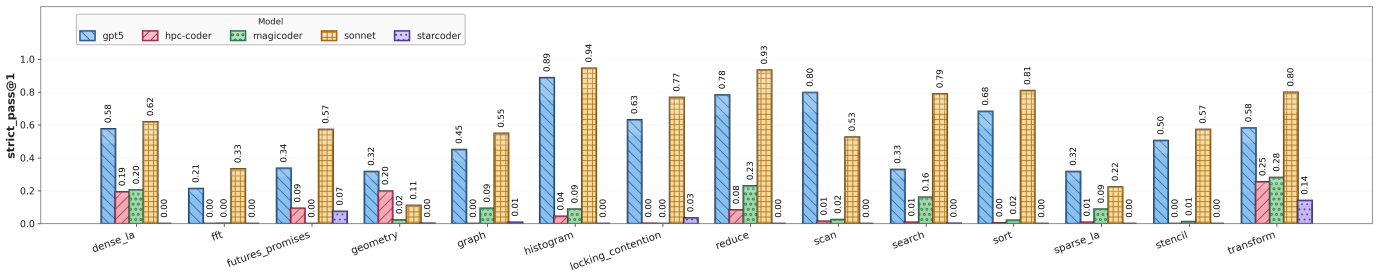


Fig. 1: strict_pass@1 (see Eqn. (1)) results for all prompts and models compiled against HPX 1.5.1 and 1.10.0

4.5 and for open models we selected Magicoder-S-DS-6.7B [12], StarCoder2-15b [13], and HPC-Coder-v2-6.7b [14].

Among closed-source models, we chose Sonnet over Opus after a preliminary comparison in which each model generated a single output per prompt in our benchmark. Sonnet was chosen since it had a higher pass@1 (0.49 vs 0.39) and more consistently used HPX namespaces, where pass@1 is the likelihood of a single code generation to pass test cases. For OpenAI models, we initially considered both GPT-5 (low reasoning) and GPT-5.1 Codex (high reasoning) on a few prompt categories of our benchmark. GPT-5 slightly outperformed GPT-5.1 Codex at 2048 tokens (our token limit) and we observed no significant difference in overall pass@1 between the two models at 4096, 8192, and 16384 token limits. Consequently, we selected GPT-5 (non-Codex) for our generation experiments. The open-source models were chosen since they achieved among the highest pass@1 among open source models on the original ParEval benchmark [14].

C. New Prompts

We added two new prompt classes with five examples each: locking/contention and futures/promises. These test LLMs on fine-grained asynchronous many-task (AMT) objectives using HPX features absent from ParEval and/or unsupported by traditional runtimes like MPI.

Locking and contention prompts evaluate whether models can (a) recognize when parallel algorithms require concurrent data access management (e.g., locks for histogram construction), and (b) implement minimal-contention solutions. Due to contention overhead, successful implementations may exhibit increased runtime with more threads and potentially underperform serial or mutex-based C++ baselines. These prompts assess whether models can implement contention-aware solutions using HPX locks.

Many AMT languages provide futures and promises for point-to-point synchronization. While ParEval's Fourier transform and graph sections could use futures, they primarily rely on synchronous parallel execution. We added a futures/promises section exercising HPX's local control objects (LCOs) for synchronization and latency reduction [15].

III. Evaluation

We evaluated all prompts and models in two stages: generation (determining source code validity based on HPX namespace usage) and execution (evaluating performance against a serial baseline) We conducted all experiments on the Unity cluster. The generation runs used an NVIDIA Tesla A100 GPU with 80 GB of VRAM. Due to the verbosity of open-source models, which often regurgitate the prompts or leave repeated blank function signatures, we cleaned each output by extracting the first generated function. The closed-source models reliably output only the requested function without extraneous repetition, indicating prompt comprehensibility. Since the new AMT-specific prompts may contain functions in the input scaffolding, these prompts are parsed specifically for the requested function signature. The execution runs used nodes with an AMD EPYC 7763 64-Core Processor and 500 GB of RAM, and we ran the cleaned code with thread counts ranging from 1 to 64 in powers of two. While HPX does support multi-node and GPU contexts, we focus our initial evaluation on single node performance. HPX does support a CUDA language extension [16] but requires calling CUDA manually, not unlike MPI+CUDA and other paradigms. Models were not asked to generate GPU (CUDA) code/kernels and did not spontaneously generate such un-prompted.

A. Generation

Generation was limited to 2048 output tokens, using a temperature (randomness) of 0.2 and a top-p sampling ratio (creativity) of 0.95 following prior work on code LLMs that observed code generation stability with these parameters [17]. GPT-5 doesn't allow temperature nor top-p controls and we set its generations to low reasoning and low verbosity to maximize token economy; with medium or high reasoning levels, the model frequently failed to produce a response due to excessive reasoning token consumption. Claude Sonnet 4.5 only allows the top-p parameter, and generations were similarly limited to 1200 reasoning tokens to best ensure complete generation outputs with adequate reasoning. We measure both GPU memory utilization and generation throughput for each experiment.

B. Execution

For each generated output, we compile and execute the code using HPX version 1.5.1 and 1.10.0, since these two versions collectively cover the majority of HPX namespace variations. An output is considered valid if it compiles, then executes and passes all test cases within 60 seconds for at least one of the two HPX versions. Invalid outputs are discarded during the efficiency calculations below, as in [6].

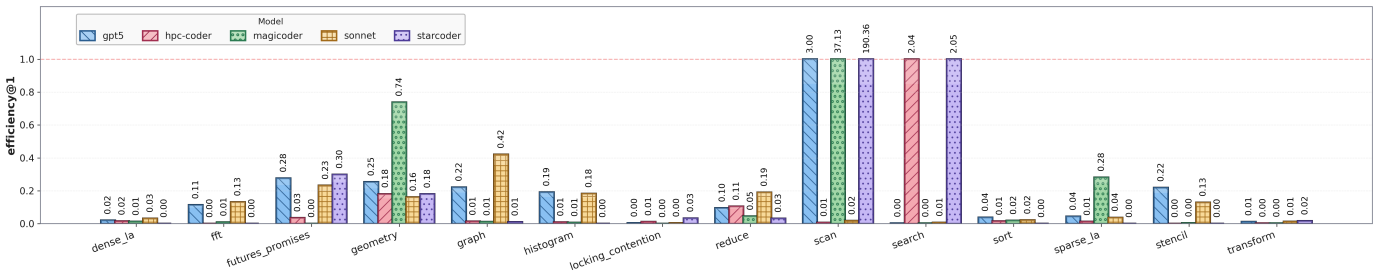


Fig. 2: efficiency@1 results for 64 threads for all prompt categories and models compiled against HPX 1.5.1 and 1.10.0

Let P being the set of problems in a prompt category for Equations (1) and (2). The `strict_pass@1` metric measures the likelihood of one code generation to pass test cases by using HPX:

$$\text{strict\_pass@1} = \frac{1}{|P|} \sum_{p \in P} \frac{\text{\# correct code samples using HPX}}{\text{total code samples (100)}}, \tag{1}$$

The `efficiency@1` metric gives a thread-normalized speedup (we compute at 64 threads) in comparison to the serial baseline [6]:

$$\text{efficiency@1} = \frac{1}{|P|} \sum_{p \in P} \sum_{j=1}^{n} \frac{\text{serial baseline runtime}}{n \cdot t_j \cdot 64}, \tag{2}$$

where n = total generations (100), t_j = j-th fastest runtime

Figure 1 shows the strict_pass@1 and Figure 2 reports efficiency@1 per prompt category.

IV. Results

Our evaluation uncovers a substantial gap between closed-source and open-source models in their ability to generate HPX code, particularly for ATM-specific patterns. Closed-source systems consistently succeed on the AMT-specific prompts, while open-source models often fail to use HPX or generate efficient code that finishes within the time limit (Fig. 1). An exception is when StarCoder got a pass@1 of 0.36 on a futures/promises prompt by correctly using HPX dataflow constructs in a tiled algorithm, resulting in the highest efficiency within this category (Fig. 2). HPC-Coder and StarCoder appear to have some familiarity with futures/promises syntax, but their outputs are significantly less likely to build nor run within the time limit in such AMT-specific categories.

The open-source models struggle most with the search, scan, sort, fft, and stencil categories (Fig. 1). Nevertheless, several models achieve high efficiency in the search and scan categories by writing conventional C++ code that outperform the serial baseline, i.e. for a scan prompt StarCoder and Magicoder devise an O(n) solution while the baseline is O(n^2)). In contrast, GPT-5 and Sonnet typically employ standard HPX constructs. In the search category, some StarCoder and HPC-Coder generated functions immediately return a boolean that happens to be correct, resulting in artificially high efficiencies.

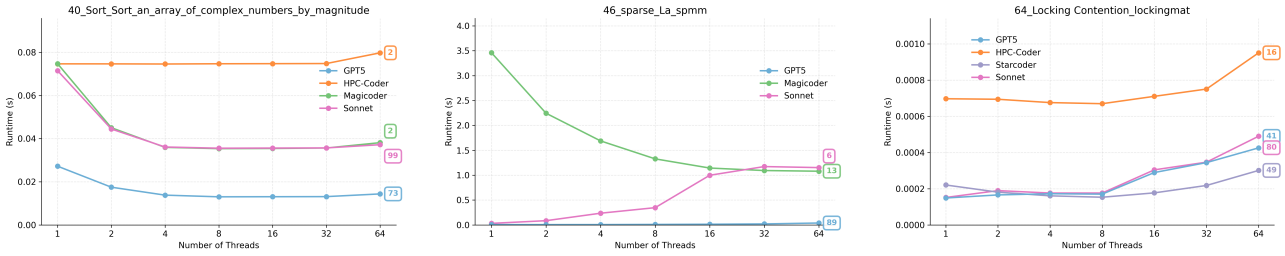
GPT-5 makes efficient operational choices that enable it to regularly scale best at 64 threads (Fig. 2). For example, it has the lowest runtime on a sorting problem by avoiding expensive square root calculations (Fig. 3a). Sonnet often follows similar scaling patterns as GPT-5 but often has a higher runtime due to inefficient choices, such as locks for a sparse matrix multiplication prompt while GPT-5 uses HPX parallel loop execution (Fig. 3b). Magicoder has very high runtime due to unnecessary parallel loops (Fig. 3b). There are instances where an open-source model achieves the lowest runtime, such as a locking/contention prompt where StarCoder is fastest by using standard library locks or pairing locks with HPX parallel loops (Fig. 3c). However, HPC-Coder has the highest runtimes from using very coarse-grained locks.

Across all benchmarks, Sonnet achieves the highest strict_pass@1 score of 0.61, followed by GPT-5 at 0.53. StarCoder performs poorest with a strict_pass@1 of 0.02 (pass@1 of 0.07). While the training corpora of GPT-5 and Claude Sonnet are proprietary, their consistent success on AMT-specific prompts suggests ample exposure to parallel and asynchronous programming concepts during pretraining.

V. Conclusions and Future Work

This study establishes a baseline comparison of closed-source and open-source LLMs for generating AMT code. Large proprietary models remain dominant in accuracy, but smaller systems such as HPC-Coder offer attractive generation throughput, with an average of 5.1 seconds per generation output, and GPU memory usage efficiency, just 46% on average.

Our long-term objective is to build a parallel-programming assistant capable of translating between existing paradigms and generating new code for under-resourced AMT frameworks. While this study has focused solely on single-node HPX generation, we are currently evaluating the ability of LLMs to generate AMT code for a variety of models, i.e. Chapel, Charm++, and Legion, and the scalability of the generated code when run across multiple nodes of execution. Key next steps include expanding the ParEval benchmark to additional tasks to identify weaknesses with closed-source models, namely by incorporating even lower-resource AMTs such as Charm++. We will evaluate larger ($\sim$100B parameters) open-source models such as OpenAI's gpt-oss and Meta's various Llama models. We plan on exploring learning strategies to improve open-source model performance.



(a) (b) (c)

Fig. 3: Average runtime with respect to the number of threads among models for selected prompts, with the total number of generations that pass all test cases (regardless of strict pass) on the right hand side.

It is a surprise that open source models were able to solve some futures/promises prompts and produce locking/contention code that built (but were too inefficient or didn't use HPX). These prompts provide more input scaffolding and data structures than the rest of ParEval, which may sometimes implicitly teach models the relevant syntax and hint usage patterns. This highlights the need for future work on prompt-engineering and in-context learning strategies, particularly for open-source models, to better process long inputs and use complex concepts. Ultimately, this line of work aims to lower the barrier for scientists and developers to adopt low-resource parallel programming tools best suited to their problems and prompt models to generate reliable outputs.

Acknowledgments

The authors would like to thank Nargiz Akhmetova, Nichole Etienne, Danielle Justo, and Ramsha Rauf for their invaluable assistance and the Unity Cluster (at MGHPCC) for computational resources and technical support. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

[1] A. P. Humphrey, "Scalable asynchronous many-task runtime solutions to globally coupled problems," Ph.D. dissertation, The University of Utah, 2019.

[2] J. Yan, H. Kaiser, and M. Snir, "Understanding the communication needs of asynchronous many-task systems – a case study of hpx+lci," 2025.

[3] B. Acun, A. Gupta, N. Jain, A. Langer, H. Menon, E. Mikida, X. Ni, M. Robson, Y. Sun, E. Totoni *et al.*, "Parallel programming with migratable objects: Charm++ in practice," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 647–658.

[4] H. Kaiser, P. Diehl, A. S. Lemoine, B. A. Lelbach, P. Amini, A. Berge, J. Biddiscombe, S. R. Brandt, N. Gupta, T. Heller, K. Huck, Z. Khatami, A. Kheirkhahan, A. Reverdell, S. Shirzad, M. Simberg, B. Wagle, W. Wei, and T. Zhang, "Hpx - the c++ standard library for parallelism and concurrency," *Journal of Open Source Software*, vol. 5, no. 53, p. 2352, 2020.

[5] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken, "Legion: Expressing locality and independence with logical regions," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.

[6] D. Nichols, J. H. Davis, Z. Xie, A. Rajaram, and A. Bhatele, "Can large language models write parallel code?" in *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '24, jun 2024.

[7] P. Valero Lara, A. Young, J. S. Vetter, Z. Jin, S. Pophale, M. Alaul Haque Monil, K. Teranishi, and W. F. Godoy, "Chathpc: Building the foundations for a productive and trustworthy ai-assisted hpc ecosystem," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2025, pp. 458–474.

[8] C. R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, and J. Wilke, "Kokkos 3: Programming model extensions for the exascale era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 805–817, 2022.

[9] D. Beckingsale, T. R. W. Scogland, J. Burmark, R. D. Hornung, H. E. Jones, W. Killian, A. J. Kunen, O. Pearce, P. Robinson, and B. S. Ryujin, "Raja: Portable performance for large-scale scientific applications," *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pp. 71–81, 2019.

[10] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[11] W.-L. Chiang, L. Zheng, Y. Sheng, A. N. Angelopoulos, T. Li, D. Li, B. Zhu, H. Zhang, M. Jordan, J. E. Gonzalez, and I. Stoica, "Chatbot arena: An open platform for evaluating LLMs by human preference," in *Forty-first International Conference on Machine Learning*, 2024.

[12] Y. Wei, Z. Wang, J. Liu, Y. Ding, and L. Zhang, "Magicoder: Empowering code generation with OSS-instruct," in *Proceedings of the 41st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 235. PMLR, 21–27 Jul 2024, pp. 52 632–52 657.

[13] BigCode, "Starcoder2-15b," https://huggingface.co/bigcode/starcoder2-15b, 2024, code large language model; Hugging Face model card for `bigcode/starcoder2-15b`.

[14] A. Chaturvedi, D. Nichols, S. Singh, and A. Bhatele, "Hpc-coder-v2: Studying code llms across low-resource parallel languages," in *ISC High Performance 2025 Research Paper Proceedings (40th International Conference), Hamburg, Germany, June 10-13, 2025*. Prometeus GmbH / IEEE, 2025, pp. 1–14.

[15] H. Kaiser, T. Heller, B. Adelstein-Lelbach, A. Serio, and D. Fey, "Hpx: A task based programming model in a global address space," in *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, ser. PGAS '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2676870.2676883

[16] M. Stumpf, M. Seshadri, P. Diehl, H. Kaiser, T. Heller, D. Howard, J. Biddiscombe, L. Viklund, and O. Katz, "Stellar-group/hpxcl: Update cmake and port to hpx 1.5.1," Apr. 2021.

[17] A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei, T. Liu, M. Tian, D. Kocetkov, A. Zucker, Y. Belkada, Z. Wang, Q. Liu, D. Abulkhanov, I. Paul, Z. Li, W.-D. Li, M. Risdal, J. Li, J. Zhu, T. Y. Zhuo, E. Zheltonozhskii, N. O. O. Dade, W. Yu, L. Krauß, N. Jain, Y. Su, X. He, M. Dey, E. Abati,

Y. Chai, N. Muennighoff, X. Tang, M. Oblokulov, C. Akiki, M. Marone, C. Mou, M. Mishra, A. Gu, B. Hui, T. Dao, A. Zebaze, O. Dehaene, N. Patry, C. Xu, J. McAuley, H. Hu, T. Scholak, S. Paquet, J. Robinson, C. J. Anderson, N. Chapados, M. Patwary, N. Tajbakhsh, Y. Jernite, C. M. Ferrandis, L. Zhang, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries, "Starcoder 2 and the stack v2: The next generation," 2024.