RADemics

Python Data Structures and Algorithms for Efficient AI Computations

Nita Thakare, Sowmya V, A. Regita Thangam
PRIYADARSHINI COLLEGE OF ENGG, SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, ST. XAVIER'S COLLEGE

Python Data Structures and Algorithms for Efficient AI Computations

[1]Nita Thakare, Associate Professor, Computer Technology, Priyadarshini College of Engg, Nagpur, Mail ID: tnita1341@gmail.com , Mobile No: 99403 64303.

[2]Sowmya V, Assistant professor, Computer Applications - Data science, SRM Institute of Science and technology, Ramapuram, Chennai, Mail ID: sowkvdev@gmail.com , Mobile No: 824 800 2831.

[3]A. Regita Thangam, Asst Professor & Head, Dept of Computer Applications, St.Xavier's College, Palayamkottai, Mail ID: regitadja@gmail.com ,Mobile No: 824 800 2831.

Abstract

The exponential growth of data and the increasing complexity of artificial intelligence (AI) models have created an urgent demand for computational efficiency, scalability, and optimized resource utilization. This book chapter presents a comprehensive exploration of how Python's data structures and algorithmic paradigms can be strategically applied to enhance the performance of AI systems. It investigates both core and advanced data structures—including lists, dictionaries, heaps, graphs, tries, Bloom filters, and sparse matrices—highlighting their computational characteristics and practical relevance in AI applications. The chapter further explores algorithmic strategies such as recursion, dynamic programming, and divide-and-conquer, illustrating their impact on processing speed, memory efficiency, and problem-solving capability within intelligent systems. Emphasis is placed on case-based performance benchmarking, revealing how structure-aware algorithm design influences real-world AI outcomes in fields such as natural language processing, deep learning, and graph analytics. The integration of high-performance Python libraries such as NumPy, SciPy, Dask, and Pandas is also examined, demonstrating how they enable scalable, parallel, and memory-optimized computations. Through a synthesis of theoretical insights and empirical evaluations, this chapter establishes a structured methodology for aligning data structure selection and algorithmic efficiency with the operational goals of modern AI systems. The findings contribute significantly to the development of scalable, interpretable, and resource-conscious AI architectures.

Keywords: Python programming, data structures, algorithmic optimization, AI scalability, parallel computing, memory efficiency

Introduction

The emergence of large-scale artificial intelligence (AI) applications has transformed the computational landscape, placing significant emphasis on the importance of efficiency, speed, and scalability [1]. With the proliferation of data-intensive models in areas such as machine learning, deep learning, and natural language processing, there is a growing demand for systems capable of processing vast volumes of information in real-time [2]. This demand necessitates the strategic selection and optimization of data structures and algorithms [3]. Python, known for its simplicity

and rich ecosystem, has become the preferred language in AI development [4]. Achieving optimal computational performance within AI pipelines requires more than syntactic convenience; it involves leveraging Python’s internal data organization models and algorithmic capabilities in a methodical manner. This chapter focuses on how effective utilization of Python’s data structures can significantly impact AI model training, inference, and overall resource consumption [5].

Understanding the underlying mechanisms of Python's data structures is vital to achieving computational efficiency [6]. Structures such as lists, tuples, sets, dictionaries, stacks, and queues form the basis of data manipulation and storage in AI workflows [7]. Each structure has unique characteristics in terms of memory usage, lookup time, and mutability, which influence performance outcomes in algorithmically intensive environments [8]. For instance, dictionaries offer constant-time average-case access, making them suitable for label mapping or embedding indexing in natural language tasks. Conversely, lists, although flexible, can incur overhead in insertion and deletion operations [9]. The choice between these structures must be aligned with task requirements, especially when dealing with high-frequency operations. A clear understanding of time and space complexities becomes crucial when designing components of AI systems where performance trade-offs are involved [10].

Beyond basic structures, advanced and custom data representations have proven to be instrumental in achieving scalability in AI solutions [11]. Data structures such as graphs, trees, tries, Bloom filters, heaps, and sparse matrices provide specialized solutions for tasks involving hierarchical relationships, probabilistic querying, and high-dimensional data [12]. For example, tries are highly efficient for autocomplete engines and NLP applications requiring prefix-based searching, while Bloom filters are used in recommendation engines to perform fast membership checks with minimal memory usage [13]. Sparse matrices are especially useful in areas like computer vision or collaborative filtering, where data is inherently sparse and memory optimization is critical [14]. Incorporating these structures into AI pipelines demands a deep understanding of their algorithmic behavior and suitability for parallel and distributed computing environments [15].