Software Metrics Analysis Report (Assignment 1)

Lirong Yi

September 30, 2025

Abstract

This report details the design and application of a software measurement instrument created to analyze large-scale, open-source projects. The toolkit was used to analyze three distinct repositories: the Linux kernel (C/C++), Apache Kafka (Java), and OpenStack Nova (Python). Four key metrics were measured: Physical and Logical Lines of Code (pLOC/lLOC), McCabe Cyclomatic Complexity (CCN), and Fan-in/Fan-out.

1 Introduction

Software metrics provide a quantitative basis for understanding and managing the complexity, size, and quality of software projects. The primary objective of this project was to build a reusable measurement instrument capable of automating the collection of four fundamental software metrics. This instrument was then applied to three open-source projects to analyze and compare their structural properties.

The repositories selected for this study are:

- **Linux Kernel**: A monolithic operating system kernel written primarily in C/C++.
- **Apache Kafka**: A distributed event streaming platform written in Java.
- **OpenStack Nova**: A cloud computing fabric controller written in Python.

2 Metric Definitions

Table 1: Toolchain for Metric Collection

Metric	**Tool Used**
Physical LOC (pLOC)	`cloc`
Logical LOC (lLOC)	`lizard`
McCabe CCN	`lizard`
Fan-in & Fan-out (C++/Java)	`CodeQL`
Fan-in & Fan-out (Python)	`analyze_python_faninout.py`

Physical LOC (pLOC) Total number of lines in the file, including code and comments, excluding blank lines. Measured by `cloc`.

Logical LOC (lLOC) Number of executable statements, ignoring formatting and line breaks. Measured by `lizard`. The tool's counting rules specify that multiple statements on a single line are counted as 1 lLOC, and a single statement spanning multiple lines is also counted as 1 lLOC.

McCabe Cyclomatic Complexity (CCN) Number of independent paths through a function; measures the complexity of control flow. Measured by `lizard`.

Fan-in Number of distinct functions that call a given function (incoming dependencies). Measured by `CodeQL` (for C++/Java) and a custom script (for Python).

Fan-out Number of distinct functions called by a given function (outgoing dependencies). Measured by `CodeQL` (for C++/Java) and a custom script (for Python).

3 Results

This section presents the visualized metrics for each repository.

Table 2: Overall Metrics Across Repositories

Metric	Linux (C)	Kafka (Java)	Nova (Python)
Total pLOC	38,203,749	1,788,815	742,220
Total lLOC	14,817,714	743,090	352,783
Average CCN	4.22	1.72	1.79
Average Fan-in	7.51	23.26	3.42
Average Fan-out	6.32	14.08	4.00

3.1 Linux Kernel (C/C++)

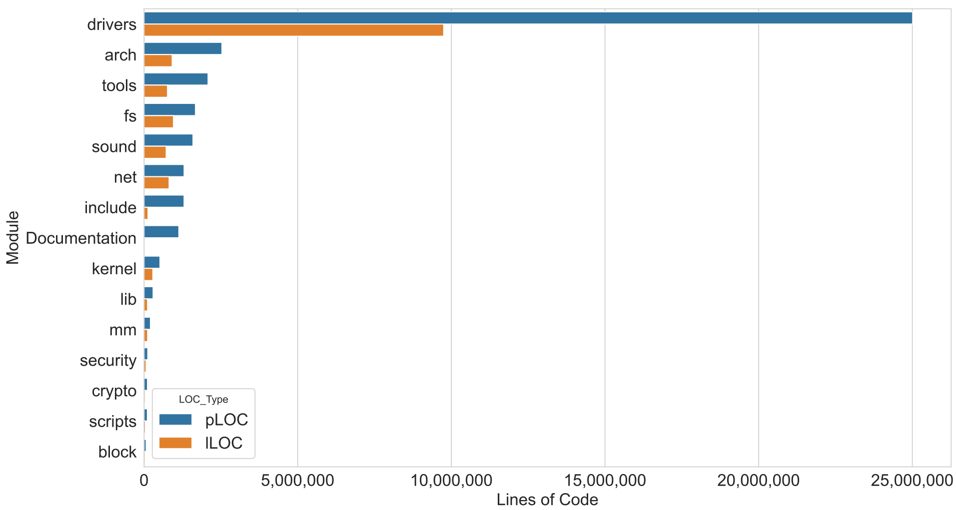

Figure 1: pLOC vs. lLOC for Top 15 Linux Modules.

Analysis: The `drivers` module is, by a significant margin, the largest module. This is expected, as it contains code to support a vast ecosystem of hardware devices.

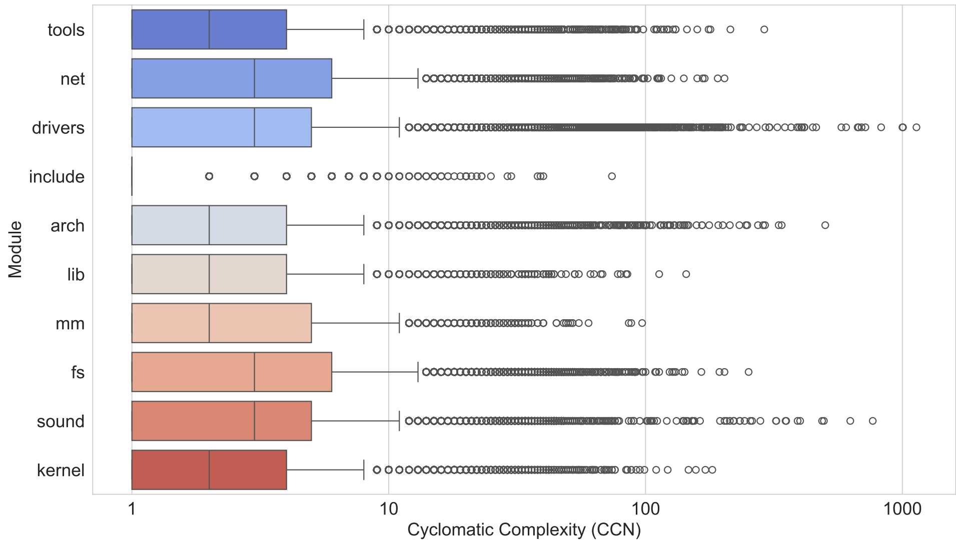

Figure 2: Distribution of McCabe Complexity (CCN) for the Top 10 largest modules.

Analysis: This box plot reveals that while most functions in the kernel have a low complexity, most modules contain a significant number of outliers with very high complexity.

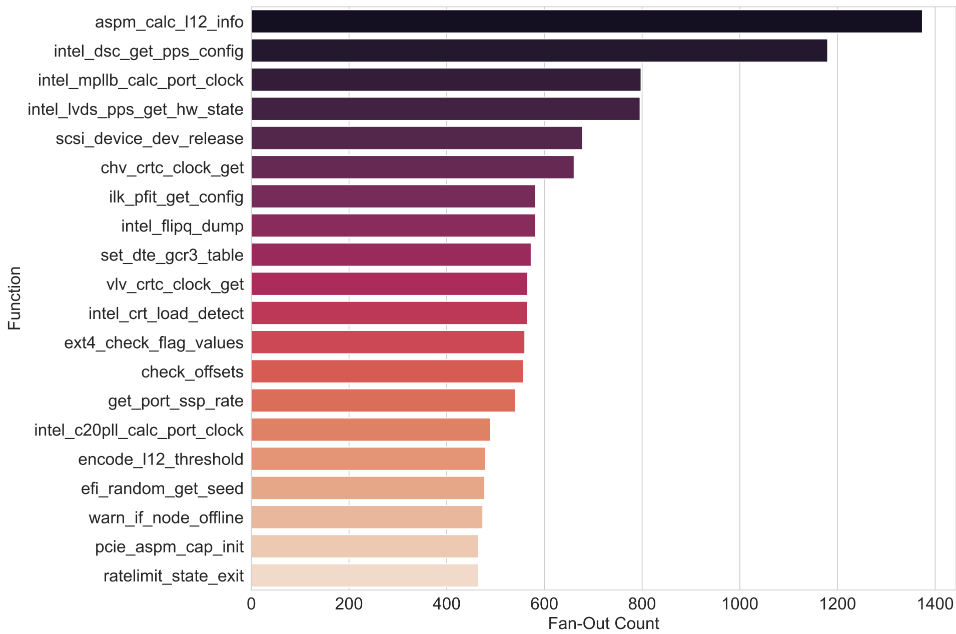

Figure 3: Top 20 functions by Fan-out.

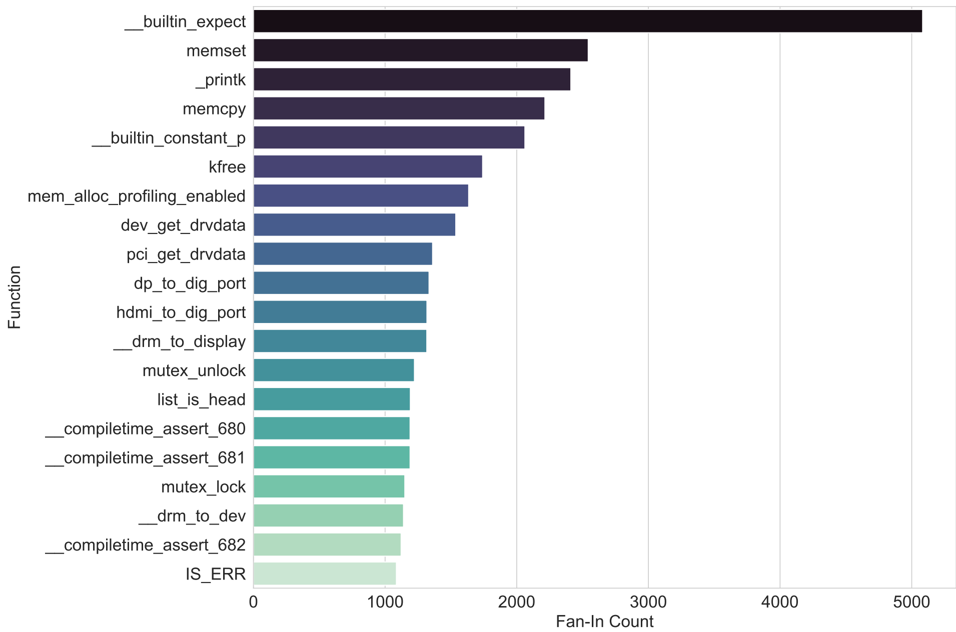

Figure 4: Top 20 functions by Fan-in.

Analysis: The functions with the highest Fan-in are foundational utilities used throughout the kernel, such as `builtin_expect`, `memset` and `printk`, confirming their central role.

3.2 Apache Kafka (Java)

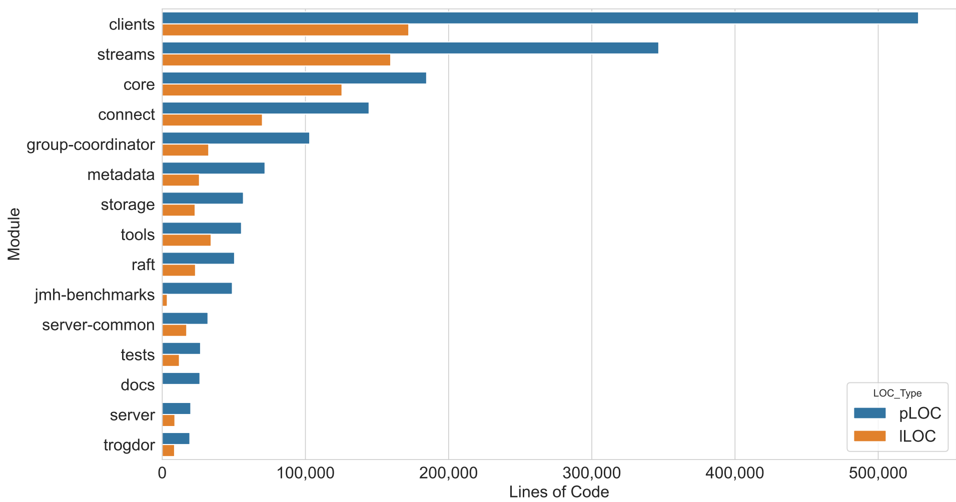

Figure 5: pLOC vs. lLOC for Top 15 Kafka Modules.

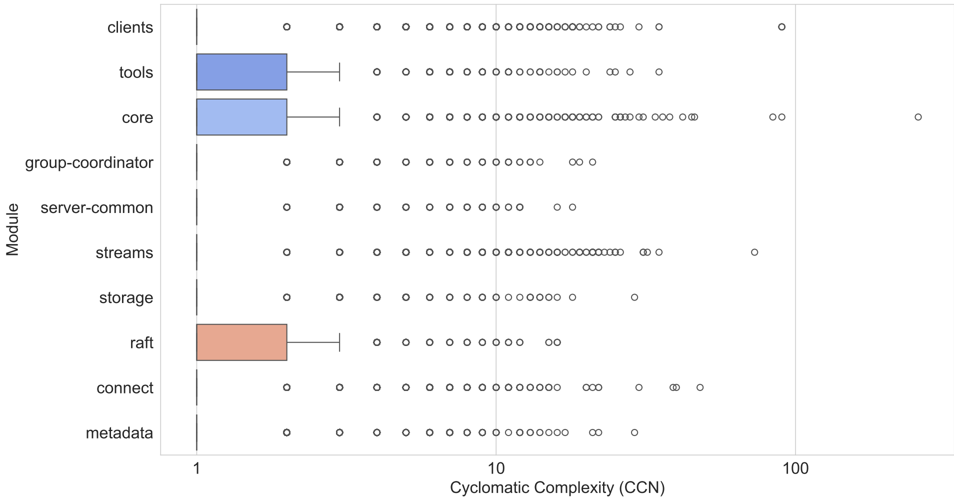

Figure 6: Distribution of CNN for the Top 10 largest modules of Kafka.

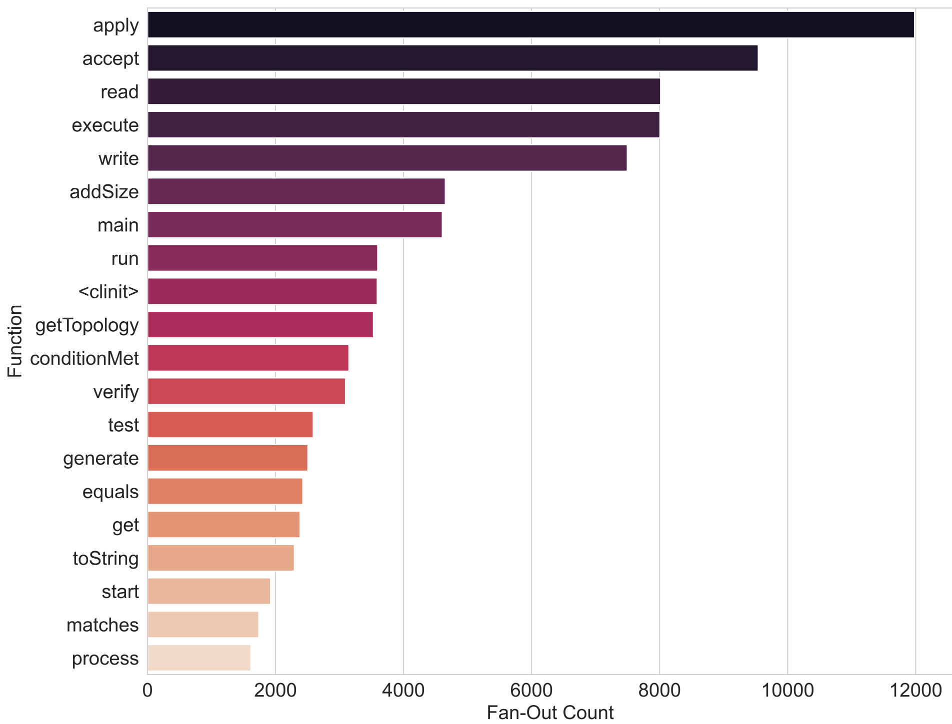

Figure 7: Top 20 functions by Fan-out.

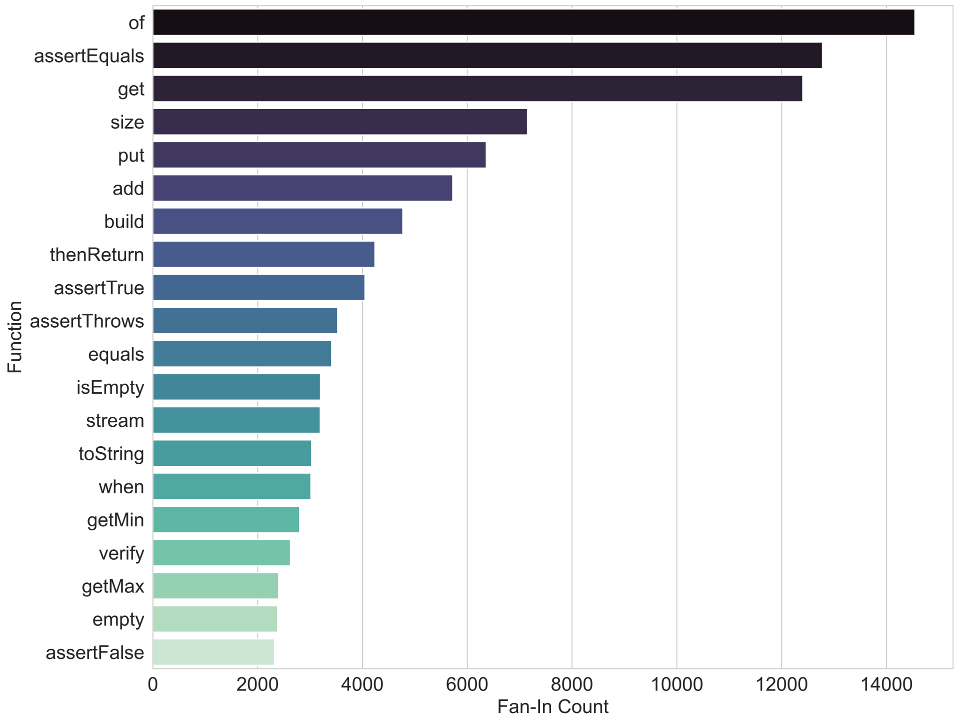

Figure 8: Top 20 functions by Fan-in.

3.3 OpenStack Nova (Python)

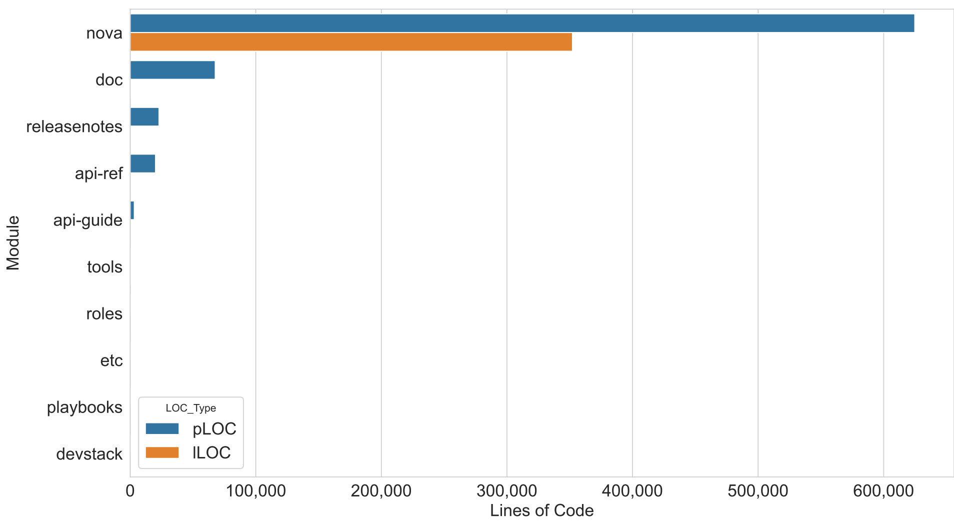

Figure 9: pLOC vs. lLOC for Top 10 Nova Modules.

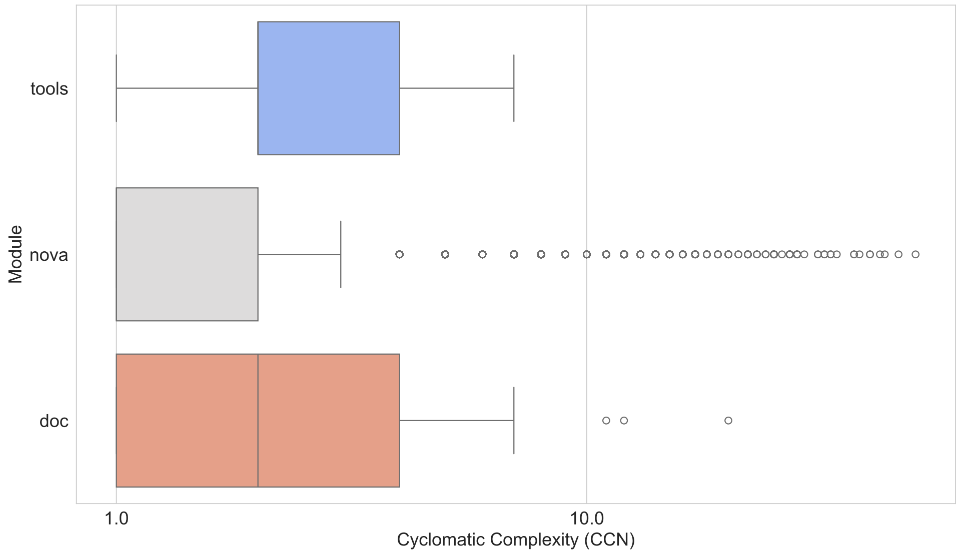

Figure 10: Distribution of CNN for the Top 3 largest modules of Nova.

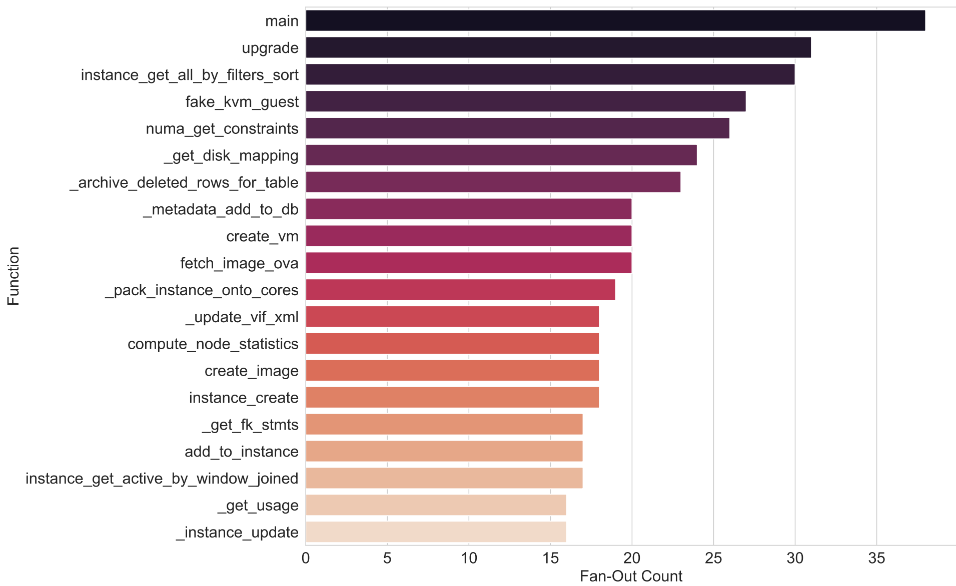

Figure 11: Top 20 functions by Fan-out.

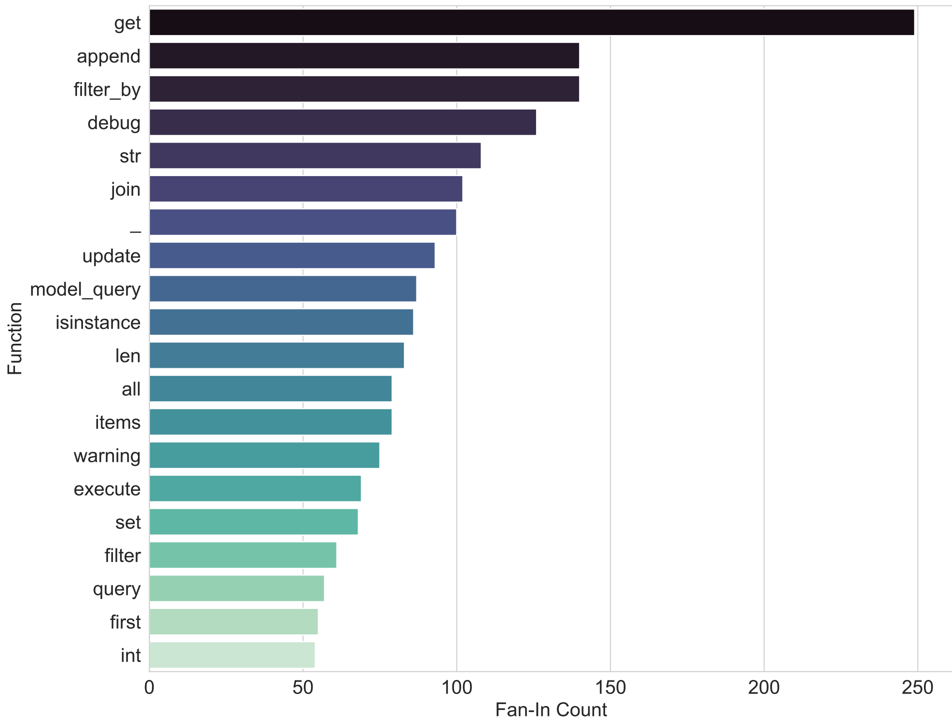

Figure 12: Top 20 functions by Fan-in.

4 Discussion & Limitations

This analysis provided valuable insights, but it is important to acknowledge its limitations.

- **lLOC Definition**: The `lizard` tool's method of counting lLOC (one per non-comment line in a function) is a useful proxy but differs from a pure statement count.
- **Fan-in/Fan-out for Python**: Statically analyzing call graphs in a dynamic language like Python is inherently challenging. The custom script provides a valuable approximation but may not capture all dynamic calls.