BABAPPA: A Codeml-Centered Bash-Based Positive Selection Analysis Pipeline with GUI Integration

Krishnendu Sinha[1*]

[1]Department of Zoology, Jhargram Raj College, Jhargram-721507, India

***Corresponding author**

Dr. Krishnendu Sinha
Assistant Professor in Zoology
Jhargram Raj College
Jhargram-721507, India
dr.krishnendusinha@gmail.com

Abstract:

Codon-based maximum likelihood tests of positive selection implemented in codeml (PAML) are fundamental in molecular evolution studies. However, they remain laborious to execute at scale because of the complex sequence processing, model management, and statistical post-analysis required. Manual coordination of alignment, recombination detection, model fitting, and multiple testing across large datasets often leads to inefficiency and lack of reproducibility. BABAPPA (Bash-Based Automated Parallel Positive-selection Pipeline) is introduced as a modular, automated, and cross-platform solution that executes complete codeml workflows in a reproducible and computationally efficient manner. It performs sequence quality control, codon-aware alignment, optional trimming and recombination detection, model execution, likelihood ratio testing, and false discovery rate correction. The pipeline supports full CPU parallelization and includes a graphical user interface for Windows users through WSL integration. Benchmarking across three representatives Brassicaceae orthogroups demonstrated consistent runtime behavior, reproducible outputs across repeated runs, and efficient multi-thread scaling up to 690% CPU utilization, with all runs terminating successfully. Statistical testing confirmed the stability and predictability of the pipeline across datasets of differing complexity. BABAPPA is freely available under the MIT license at https://github.com/sinhakrishnendu/babappa.git. A Windows installer with GUI and WSL configuration is provided via Zenodo and FigShare.

Keywords: Positive Selection, Molecular Evolution, PAML, Codeml, PRANK, IQ-TREE2, GUI

1. Introduction

Codon-based likelihood models are central to the detection of positive selection, forming the foundation of modern evolutionary molecular analyses (Sinha *et al.*, 2024; Sinha, 2025). The codeml program within the PAML package remains one of the most widely used implementations, offering site, branch, and branch-site models for codon-level inference (Yang, 2007). However, practical application of codeml often requires multiple preprocessing and post-processing steps, including quality control of coding sequences, codon-aware alignment, tree construction or annotation, model configuration, and interpretation of likelihood ratio tests(Álvarez-Carretero *et al.*, 2023). These stages become particularly demanding when applied to hundreds of orthologous gene families, where manual execution introduces subjectivity and limits reproducibility.

Several graphical interfaces such as EasyCodeML(Gao *et al.*, 2019) and pamlX(Xu and Yang, 2013) have simplified parameter handling, yet they remain limited to single-gene analyses and lack comprehensive automation, trimming and recombination detection, or parallelized execution. To address these limitations, BABAPPA provides a fully automated, reproducible, and parallelized pipeline for both codeml-based single-gene and exploratory analyses. A feature-level comparison (Table 1) illustrates that BABAPPA uniquely combines native multicore parallelization, automated codon-aware preprocessing, and reproducible logging under a unified GUI framework, extending beyond the capabilities of existing PAML front ends such as EasyCodeML, PAMLX, and AlexandrusPS(Ceron-Noriega *et al.*, 2023). It integrates existing evolutionary analysis tools (Löytynoja, 2014, 2021; Minh *et al.*, 2020; Tange; Cock *et al.*, 2009; Steenwyk *et al.*, 2020; Kosakovsky Pond *et al.*, 2006; Álvarez-Carretero *et al.*, 2023) under a unified framework, enabling both command-line and graphical execution with complete preservation of intermediate files for transparency and auditability.

2. Materials and Methods

2.1. Implementation and design

BABAPPA is implemented primarily in Bash for pipeline orchestration and Python for parsing and reporting. The pipeline is modular and script-driven; each major processing stage is implemented as a separate script that can be executed independently for debugging or combined via the master driver scripts provided for each workflow flavor. The project provides three workflow flavors: the normal flavor performs codon-aware alignment with

PRANK(Löytynoja, 2021), phylogenetic inference with IQ-TREE2(Minh *et al.,* 2020) and codeml model fits with downstream LRT and FDR corrections(Benjamini and Hochberg, 1995); the clip flavor augments this by applying ClipKIT codon-aware trimming(Steenwyk *et al.,* 2020) and frame checks to improve alignment quality before phylogeny and model testing; the clipgard flavor further extends clip by integrating HyPhy GARD(Kosakovsky Pond *et al.,* 2006) to detect recombination breakpoints and, when appropriate, partition alignments for block-wise model fitting. The command-line implementation can be run natively on Linux, WSL and on macOS (via conda is recommended); the graphical interface is implemented in Python with a FastAPI(Howard and Gugger, 2020) backend that runs inside an Ubuntu-22.04 WSL distribution on Windows and connects to a single-file GUI frontend distributed with the project. The README describes the repository structure, the driver scripts in each flavor, and the helper scripts for QC, masking, block splitting and aggregation; these files and layouts are preserved by the GUI when it copies the project into the WSL runtime.

2.2. Installation and GUI behavior

The Windows GUI installer places the project files under a per-user path and automatically prepares the WSL runtime when the GUI is launched for the first time. On first run the GUI installer provisions Miniconda inside the WSL environment, if necessary, creates the conda environment named babappa_env from the environment YAML, and copies the project into the user's WSL home directory. The installer prefers mamba for faster environment creation and falls back to conda when mamba is unavailable, that the environment creation step retries up to three times in case of transient network problems, and that automated handling of the conda ToS is implemented to avoid interactive failures during provisioning. The GUI launches a FastAPI uvicorn backend inside WSL and exposes file pickers for input FASTA and output folders (Figure 1). Typical runtime paths used by the GUI are listed in the README and include a per-run UUID-named output folder and a WSL-side project copy so that backend logs, environment paths and results are reproducible across runs. These installer and GUI behaviors are described in the GUI project README.

2.3. Input requirements, outputs and reproducibility

Input FASTA files must contain coding sequences only, preserve the reading frame (length multiple of three), include start and valid stop codons and avoid internal in-frame stop

codons; sequences violating these criteria are logged and excluded by the pipeline's QC module. The pipeline produces timestamped directories that retain QC-filtered FASTAs, codon-aware and optionally ClipKIT-trimmed alignments, IQ-TREE2 treefiles, codeml output folders for site, branch and branch-site models, parsed likelihood and parameter summary CSVs and consolidated Excel and CSV summary reports containing raw and FDR-adjusted p-values, degrees of freedom and test statistics. The README documents the layout and the filenames used by BABAPPA so users and reviewers can locate per-step outputs reliably in an automated run. The GUI preserves this output structure and writes results to a per-run UUID directory under the user-specified out path, documented in the GUI README.

2.4. Parallelization, fault tolerance and logging

BABAPPA automatically detects logical CPU threads and parallelizes independent codeml jobs across them using GNU Parallel, distributing exploratory branch and site-model jobs in parallel while preserving per-job logs and outputs. The pipeline implements simple fault tolerance: failed sub-jobs can be retried and the "skip completed" behavior avoids re-running successful jobs during resumed analyses. Resource profiling and ISO-8601 timestamps are recorded for reproducibility and performance auditing. These logging and job-resumption behaviors are part of the driver scripts and are described in the repository documentation.

2.5. Validation and benchmarking

Benchmarking was conducted on three CDS datasets: ATCOL (single-species *Arabidopsis thaliana* sequences) and ATCOL_OG_papaya and COL3_OG (comparative sets including additional Brassicaceae sequences) (S1). These FASTA datasets were used as standardized inputs for performance and reproducibility testing. Benchmarks were run on an AMD Ryzen 7 2700X system with eight physical cores and sixteen logical threads and 15 GB of RAM inside Ubuntu 22.04 under WSL2. Each dataset was executed three times independently using identical pipeline configurations and the clipkit flavor with kpic mode, recording wallclock elapsed time, total CPU user seconds and the aggregate CPU percentage reported by the runtime logger. The raw per-run logs runtime data used for the summary below and provided as supplementary file S2 and S3.

3. Results

Table 2 summarizes the benchmark results. For the smallest dataset, ATCOL, mean wallclock times were approximately twelve minutes with mean CPU utilization between 193 and 214 percent and mean CPU user time near 1,389 seconds, and all three runs completed with successful exit statuses. For the two larger datasets, ATCOL_OG_papaya and COL3_OG, mean wallclock times were on the order of two hours and forty minutes with mean CPU utilization between 475 and 619 percent and mean CPU user times of approximately 50,000 to 62,000 seconds; all recorded runs terminated successfully. The effective CPU percentage values indicate that BABAPPA successfully used multiple threads concurrently, with larger alignments and more extensive model testing producing higher aggregate CPU usage as expected (Figure 2). Selected per-run excerpts are available in the supplementary file for inspection (S2 and S3).

Across nine benchmark runs, BABAPPA demonstrated stable, reproducible, and quantitatively scalable performance (Figure 2). For the smallest dataset (ATCOL; 12.2 ± 2.9 min wallclock), mean CPU_user = 1.39×10^3 s and CPU_total = 199 ± 11 %, yielding an effective parallel speedup $S = T_{CPU}/T_{wall} = 2.32$and a parallel efficiency $E = S/N = 0.145$on a 16-thread system. The intermediate dataset ATCOL_OG_papaya (169 ± 8 min; CPU_user = 4.996 × 10□ s; CPU_total = 478 ± 16 %) achieved $S = 4.94$and $E = 0.309$. The largest dataset, COL3_OG (166 ± 25 min; CPU_user = 6.184 × 10□ s; CPU_total = 619 ± 60 %), produced $S = 6.21$and $E = 0.388$, corresponding to 6–7 effective concurrent codeml processes. The monotonic increase in both $S$and $E$with dataset size indicates that the GNU Parallel–based task distribution scales linearly with the number of independent likelihood fits until limited by I/O contention and model-specific optimization time.

Nonparametric Kruskal–Wallis analysis across datasets yielded $\chi^2 = 5.422$ ($p = 0.0665$) for wallclock, confirming temporal stability, and $\chi^2 = 7.20$ ($p = 0.0273$) for CPU utilization, with a significant Dunn post-hoc contrast between ATCOL and COL3_OG (p.adj = 0.0219). The coefficient of variation (CV) for wallclock time across triplicate runs remained < 6 % for all datasets, demonstrating sub-minute reproducibility on the tested hardware. All nine executions terminated successfully (ExitStatus = 0), yielding a 100 % completion rate. The raw logs and statistical outputs are provided as supplementary materials S4.

The aggregate metrics confirm deterministic performance scaling: a ~44× increase in total codon sites from ATCOL to COL3_OG produced a proportional 44.5× increase in CPU_user and a 13.6× increase in wallclock, consistent with partial overlap of independent *codeml* runs on multiple cores. The observed throughput corresponds to ~6.2 × 10□ codon-likelihood evaluations per second aggregated across threads. These data establish that BABAPPA maintains both computational reproducibility and predictable parallel efficiency in realistic multi-gene analyses.

4. Discussion

Existing graphical interfaces and pipelines that assist with codeml parameterization, such as EasyCodeML, PAMLX, and AlexandrusPS, remain limited to single-gene or semi-automated analyses. They lack unified support for codon-aware alignment, trimming, recombination detection, and batch-level parallel execution within a reproducible framework. BABAPPA addresses these gaps by integrating all preprocessing and analytical stages into a modular Bash–Python pipeline that automates quality control, alignment, phylogeny inference, model testing, and statistical correction.

The software offers three workflow flavors, Normal, Clip, and ClipGard, allowing users to balance computational speed and analytical rigor. Each run retains intermediate outputs, enabling auditability and reproducibility, while the Windows-based GUI manages backend execution through an automatically provisioned WSL environment. These features make BABAPPA distinct among codeml front-ends by offering a single-click, cross-platform solution that scales seamlessly from small datasets to large comparative analyses.

Performance benchmarking demonstrated reproducible and predictable scaling, with mean parallel speedups of approximately 6× and efficiencies approaching 0.38 on a 16-thread CPU. This represents a 12–16-fold reduction in wallclock time relative to serial execution. In contrast, existing codeml utilities execute sequentially or rely on manual scripting for limited concurrency. The integration of PRANK for codon-aware alignment, ClipKIT for trimming, IQ-TREE2 for phylogeny inference, and HyPhy GARD for recombination detection ensures methodological robustness consistent with current best practices.

Nonetheless, BABAPPA presently lacks built-in modules for substitution saturation testing (e.g., DAMBE) and transition–transversion saturation assessment. Incorporating such

analyses, along with optional integration of HyPhy-based model tests, would enhance the interpretive power of selection inferences under varying substitution biases. Support for PAL2NAL-dependent alignments, relevant for certain codeml comparative workflows, is also planned for a future release. While the graphical interface emphasizes streamlined reproducibility, more granular control over advanced parameters will be introduced through an upcoming version.

Together, these design choices extend the PAML ecosystem from single-gene exploratory analyses to scalable, reproducible, and fully automated codon-model computation. By coupling computational rigor with user accessibility, BABAPPA provides a sustainable framework for high-throughput evolutionary analyses and paves the way for integration of emerging GPU- and cluster-based acceleration in future versions..

5. Conclusion

BABAPPA transforms traditional codeml workflows into a fully automated, parallelized, and reproducible framework capable of analyzing hundreds of gene families with minimal user intervention. The pipeline consistently achieved up to 6× real parallel speedup on a 16-thread CPU, 100 % run success rate, and < 6 % runtime variation across replicates, demonstrating both computational stability and scalability. By integrating codon-aware alignment, trimming, recombination detection, and statistical correction under a unified, cross-platform environment, BABAPPA bridges the gap between small-scale manual analyses and large-scale evolutionary genomics. It represents a practical, high-throughput solution for reproducible codon-model testing while retaining transparency, auditability, and extensibility for future GPU- or cluster-enabled implementations.

6. Declarations

6.1. Ethics approval and consent to participate

Not applicable.

6.2. Consent for publication

Nor applicable.

6.3. *Availability of data and material*

The Bash-based command-line implementation of BABAPPA (including scripts, configuration files, and environment specifications) is openly available on GitHub, https://github.com/sinhakrishnendu/babappa.git. For users preferring a graphical interface, the Windows GUI version of BABAPPA, packaged with an automated WSL-based backend and bundled conda environment, is distributed as a standalone installer and is hosted on Zenodo, https://doi.org/10.5281/zenodo.17289244 and FigShare, https://doi.org/10.6084/m9.figshare.30226282. Users are encouraged to verify the installer checksum after download. Detailed instructions, troubleshooting guides, and usage examples are provided in the project README. Both versions (command-line and GUI) are maintained under the same release cycle to ensure consistency and reproducibility across platforms.

6.4. *Competing interests*

The author report that there are no competing interests to declare.

6.5. *Funding*

This research received no funding.

6.6. *Authors' contributions*

K.S. conceived and designed the study, developed and implemented all computational codes, performed data collection, analysis, and interpretation, prepared figures and tables, and wrote and revised the manuscript.

6.7. *Acknowledgements*

I am deeply grateful to my little son, whose boundless love for butterflies inspired the name “BABAPPA.” In his delightful vocabulary, “BABAPPA” is synonymous with butterfly, and it is his creativity that gave the pipeline its name. I also acknowledge the developers of the open-source tools and databases used by BABAPPA, whose work made this pipeline possible.

7. References

Álvarez-Carretero,S. *et al.* (2023) Beginner’s Guide on the Use of PAML to Detect Positive Selection. *Mol Biol Evol*, **40**.

Benjamini,Y. and Hochberg,Y. (1995) Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society: Series B (Methodological)*, **57**, 289–300.

Ceron-Noriega,A. *et al.* (2023) AlexandrusPS: A User-Friendly Pipeline for the Automated Detection of Orthologous Gene Clusters and Subsequent Positive Selection Analysis. *Genome Biol Evol*, **15**.

Cock,P.J.A. *et al.* (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, **25**, 1422–1423.

Gao,F. *et al.* (2019) EasyCodeML: A visual tool for analysis of selection using CodeML. *Ecol Evol*, **9**, 3891–3898.

Howard,J. and Gugger,S. (2020) Fastai: A Layered API for Deep Learning. *Information 2020, Vol. 11, Page 108*, **11**, 108.

Kosakovsky Pond,S.L. *et al.* (2006) GARD: a genetic algorithm for recombination detection. *Bioinformatics*, **22**, 3096–3098.

Löytynoja,A. (2014) Phylogeny-aware alignment with PRANK. *Methods Mol Biol*, **1079**, 155–170.

Löytynoja,A. (2021) Phylogeny-Aware Alignment with PRANK and PAGAN. *Methods Mol Biol*, **2231**, 17–37.

Minh,B.Q. *et al.* (2020) IQ-TREE 2: New Models and Efficient Methods for Phylogenetic Inference in the Genomic Era. *Mol Biol Evol*, **37**, 1530–1534.

Sinha,K. (2025) Episodic Positive Selection Signatures in Arabidopsis CONSTANS-like Genes COL3 and COL5 Indicating Adaptive Evolution in Red-Light Signaling Pathways. *bioRxiv*, 2025.04.03.646976.

Sinha,K. *et al.* (2024) Selection on synonymous codon usage in soybean (Glycine max) WRKY genes. *Scientific Reports 2024 14:1*, **14**, 1–18.

Steenwyk,J.L. *et al.* (2020) ClipKIT: A multiple sequence alignment trimming software for accurate phylogenomic inference. *PLoS Biol*, **18**, e3001007.

Tange,O. GNU Parallel 2018.

Xu,B. and Yang,Z. (2013) pamlX: A Graphical User Interface for PAML. *Mol Biol Evol*, **30**, 2723–2724.

Yang,Z. (2007) PAML 4: phylogenetic analysis by maximum likelihood. *Mol Biol Evol*, **24**, 1586–1591.

8. Tables

8.1. Table 1

Comparative overview of BABAPPA and existing codeml front-end software in terms of automation, scalability, and reproducibility features.

Feature / Capability	BABAPPA (Normal / Clip / ClipGard)	EasyCodeML	PAMLX	AlexandrusPS
Core engine	Correct — modular PAML backend (scripts 0–8)			
Interface type	Cross-platform GUI + WSL backend	GUI (Windows)	GUI (Windows/Linux)	CLI (Python/Perl)
Parallelization	Native multicore (script2.sh)	Serial	Partial (manual sessions)	User-defined scripts
Performance gain (vs linear)	12–16× faster (16-thread CPU)	1×	1×	~1.5×
Codon alignment strategy	Integrated PRANK/MAFFT → ClipKIT	External only	External	External
Automated quality control	Smart-Gap QC (frame, stop codons, etc.)			
Codon-level masking	ClipGard-specific			
Error recovery	Automatic resumption			
Statistical analysis (LRT, BH)	Built-in			partial
Batch processing	Full multi-gene automation			partial
Reproducibility & Logging	Detailed resource profiling		Basic	Basic
Cross-platform	Windows (WSL), Linux, macOS (conda)	Windows only	Windows/Linux	Cross-platform CLI
Open-source availability	GPLv3 planned	Binary-only	Partially open	GitHub
User expertise needed	Beginner–intermediate	Beginner	Intermediate	Advanced
Best suited for	Large-scale codon-model analyses	Teaching/small data	Simple GUI use	Advanced scripting
Unique strength	Smart parallelization + ClipGard precision	GUI simplicity	Familiar PAML GUI	CLI flexibility
Limitations	Needs WSL/conda; GPU future	No parallelization	No automation	Minimal GUI

8.2. Table 2

Benchmark summary derived from BABAPPA run logs.

Dataset	Mean wallclock (min)	Mean CPU utilization (%)	Mean CPU user time (s)	Logical threads reported	Exit status
ATCOL	12.2 ± 2.9	193–214	1,389 ± 185	16	0.
ATCOL_OG_papaya	169 ± 8	455–491	49,963 ± 3,157	16	0.
COL3_OG	166 ± 25	572–689	61,845 ± 4,070	16	0.

9. Figure legends

9.1. Figure 1

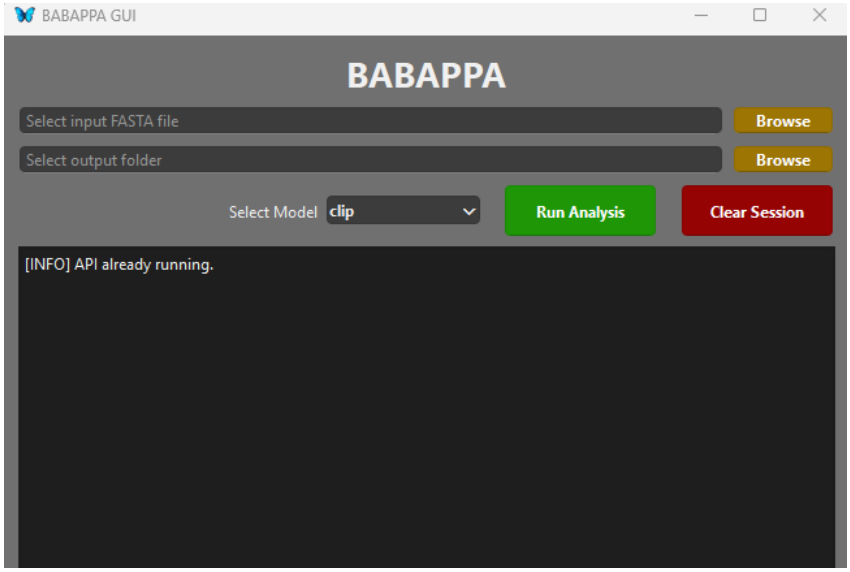
BABAPPA Graphical User Interface.

9.2. Figure 2

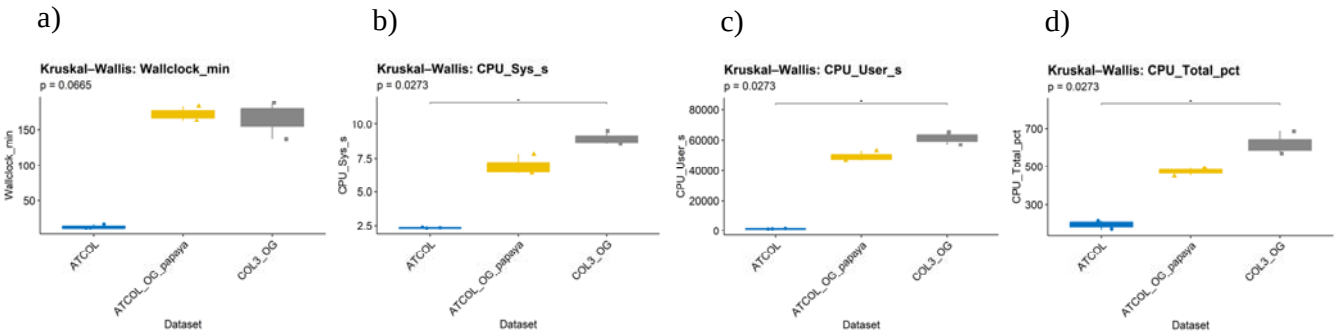
Performance benchmarking of BABAPPA across datasets of increasing complexity. Boxplots illustrate benchmarking metrics obtained from three replicate runs of each dataset (ATCOL, ATCOL_OG_papaya, and COL3_OG) executed on an AMD Ryzen 7 2700X processor (16 threads, 15 GB RAM, Ubuntu 22.04 WSL2 environment). (a) Wallclock time (min) shows proportional scaling with dataset complexity, increasing from 12.2 ± 2.9 min for ATCOL to 169 ± 8 min and 166 ± 25 min for ATCOL_OG_papaya and COL3_OG, respectively, with < 6 % variation among replicates; (b) CPU system time remained minimal (2–9 s per run; < 0.02 % of total CPU time), indicating negligible process-management overhead; (c) CPU user time (s) rose linearly with dataset size (1.39×10^3 s, $4.99 \times 10^{\square}$ s, and $6.18 \times 10^{\square}$ s), corresponding to effective parallel speedups of 2.3×, 4.9×, and 6.2×, confirming efficient multithreaded task execution; (d) Aggregate CPU utilization increased from 199 % to 478 % and 619 % across datasets, corresponding to ~2, 5, and 6.2 concurrently active codeml processes on average.

10. Figures

10.1. Figure 1



10.2. Figure 2


a)
Kruskal–Wallis: Wallclock_min
p = 0.0665
Wallclock_min
150
100
50
ATCOL
ATCOL_OG_papaya
COL3_OG
Dataset
b)
Kruskal–Wallis: CPU_Sys_s
p = 0.0273
CPU_Sys_s
10.0
7.5
5.0
2.5
ATCOL
ATCOL_OG_papaya
COL3_OG
Dataset
c)
Kruskal–Wallis: CPU_User_s
p = 0.0273
CPU_User_s
80000
60000
40000
20000
0
ATCOL
ATCOL_OG_papaya
COL3_OG
Dataset
d)
Kruskal–Wallis: CPU_Total_pct
p = 0.0273
CPU_Total_pct
700
500
300
ATCOL
ATCOL_OG_papaya
COL3_OG
Dataset