BABAPPA v2.0: A Bash-Based Automated Pipeline for Codeml-Mediated Selection Analysis with Integrated Alignment Trimming and Recombination Detection Options

Krishnendu Sinha[1*]

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

***Corresponding author**

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

Abstract:

Detecting positive selection in protein-coding genes is central to understanding adaptive evolution, yet conventional workflows based on codeml (PAML) demand extensive manual configuration and sequential execution, limiting their scalability. BABAPPA v2.0 (Bash-Based Automated Parallel Positive Selection Analysis) has present, a fully automated pipeline that streamlines all stages of codeml analysis. In contrast to v1.0, the updated release introduces three flavours of execution: BABAPPA_normal for the original streamlined workflow, BABAPPA_clip which integrates ClipKIT-based alignment trimming with codon integrity checks, and BABAPPA_clipgard which further incorporates recombination filtering via HyPhy GARD. These additions provide flexibility for balancing speed, stringency, and robustness while retaining full automation. These improvements make BABAPPA the only available framework to seamlessly automate episodic (branch-site) selection analyses alongside classical site and branch models. Validation on three Brassicaceae ortholog sets confirmed that BABAPPA v2.0 reproduces the exact positive-selection calls of a conventional manual workflow (after FDR correction), while cutting runtimes nearly in half. Mean wall-clock time dropped from 2,579 s to 1,430 s, a 44.6% reduction equivalent to an 80.3% speedup ($p = 8.227\times10^{\square\square}$). These gains are achieved without loss of sensitivity. By combining reproducibility, efficiency, and expanded model coverage, BABAPPA v2.0 provides the first practical end-to-end solution for large-scale genome-wide surveys of adaptive evolution.

Keywords: Positive Selection, Molecular Evolution, PAML(Codeml), PRANK, IQ-TREE2, Phylogenetics

1. Introduction

The detection of positive (adaptive) selection in protein-coding genes is a cornerstone of molecular evolution studies, as it reveals adaptive changes across lineages[1,2]. Maximum-likelihood models implemented in PAML's codeml (including site, branch, and branch-site models) remain a gold standard for such detection[3]. However, traditional codeml workflows are labor-intensive: researchers must manually edit configuration files, run multiple model comparisons sequentially, and collate likelihood-ratio tests (LRTs) across many genes and branches, applying corrections for multiple testing[4]. These manual steps are prone to error and severely limit throughput when analyzing large gene families or set of orthologs/paralogs. To overcome these bottlenecks, BABAPPA has been developed, a bash-based automated pipeline that orchestrates the entire codeml analysis from raw sequences to statistically validated results. BABAPPA automates sequence quality control, codon-aware alignment, phylogenetic inference, branch annotation, model fitting, and LRT-based selection testing, all with parallel execution to maximize efficiency. It leverages standard bioinformatics tools, such as Biopython[5] for sequence handling, PRANK[6] for codon alignment, IQ-TREE2[7] for phylogenetic inference, and GNU Parallel[8] for parallel processing alongside custom scripting.

In its current version, BABAPPA is available in three complementary flavors. The classic workflow, referred to as babappa_normal, performs codon alignment, tree reconstruction, and codeml-based detection of selection. A second variant, babappa_clip, incorporates ClipKIT[9] for alignment trimming with user-selectable models such as smart-gap or kpic etc., followed by automated quality control to ensure that reading frames are preserved and no premature stop codons are introduced during the trimming process. A third variant, babappa_clipgard, extends babappa_clip by integrating recombination breakpoint detection using HyPhy GARD[10], thereby addressing the well-known confounding effect of recombination on positive selection inference.

Together, these extensions enhance both the robustness and reliability of selection analyses, providing users with flexible, reproducible workflows that minimize manual intervention while improving biological interpretability (Figure 1). Here, the design of BABAPPA is described, and its performance is demonstrated against a standard codeml-based pipeline.

2. Materials and Methods

2.1. Validation datasets

To evaluate the accuracy, efficiency, and reproducibility of BABAPPA, three orthologous gene groups from Brassicaceae species has been selected. These gene sets were obtained from the PlantTFDB, a curated database of plant transcription factors. Each orthogroup comprised five putative orthologs drawn from three closely related Brassicaceae species, representing a realistic yet computationally tractable test case. The use of transcription factor families ensures inclusion of genes typically studied in codeml-based evolutionary analyses, as these genes often exhibit lineage-specific divergence and potential signatures of adaptive evolution. Gene identifiers and sequence details for each orthogroup are provided in Supplementary File S1, enabling independent replication of our tests. This dataset design reflects a typical use scenario for BABAPPA: moderate-size orthogroups spanning related species where alignment quality, recombination, and branch-specific signals of selection must all be carefully assessed.

2.2. Pipeline implementation

BABAPPA is implemented as a suite of modular Bash and Python scripts coordinated by a master driver script (`babappa_normal/clip/clipgard.sh`). The pipeline executes an end-to-end workflow that begins with raw nucleotide coding sequences (CDS) and concludes with statistically validated selection results summarized in human-readable formats (Figure 1). The modular design allows users to run individual steps if desired, while the master driver ensures complete automation for batch-scale analyses. In its current release, BABAPPA supports three complementary flavors. The babappa_normal mode reproduces the classic codeml workflow with automated configuration, execution, and aggregation. The babappa_clip mode extends this by incorporating alignment trimming with ClipKIT[9], followed by codon-aware quality control to ensure that no reading frames are disrupted during trimming. Finally, the babappa_clipgard mode builds on babappa_clip by incorporating recombination breakpoint detection using HyPhy GARD[11], addressing the confounding influence of recombination on positive selection inference. This flexible design allows users to choose between speed, robustness, or recombination-aware analysis depending on their research objectives.

2.2.1 Sequence Quality Control

The first step of the pipeline ensures that only biologically meaningful and codon-compatible sequences are retained. A dedicated Python module (`seqQC.py`) applies stringent codon-level filters following best-practice recommendations[1]. Specifically, sequences must begin with an ATG start codon and end with a valid termination codon (TAA, TAG, or TGA). They must be at least >300 base pairs in length to minimize spurious results from very short sequences, contain only the canonical nucleotides A, T, G, and C, and be free of internal in-frame stop codons. In addition, extreme outliers in sequence length (outside 1.5× the interquartile range) are automatically discarded, as such sequences often represent pseudogenes, partial transcripts, or annotation errors. This step ensures that downstream alignment and evolutionary modeling are based on sequences that are structurally and functionally interpretable as protein-coding genes.

2.2.2 Phylogeny-Aware Alignment

High-quality sequences passing initial QC are aligned using PRANK[6] in codon mode (via `script0.sh`). Unlike heuristic aligners, PRANK employs a phylogeny-aware algorithm that considers insertion/deletion events in an evolutionary context. This is particularly important for positive selection analyses, since misplaced indels or misaligned codons can produce false signals of selection. Alignments produced by PRANK therefore preserve codon structure and improve the reliability of downstream likelihood ratio tests. In babappa_normal, these alignments proceed directly to phylogenetic inference, whereas in babappa_clip and babappa_clipgard, they undergo additional refinement through trimming and validation (see below).

2.2.3 Alignment Trimming and Frame Validation (babappa_clip and babappa_clipgard only)

Alignment trimming removes ambiguous or poorly supported regions from multiple sequence alignments. In the babappa_clip and babappa_clipgard workflows, PRANK-generated alignments are processed with ClipKIT[9], which trims poorly aligned or phylogenetically uninformative sites in a codon-aware mode. Users can choose the trimming model most appropriate for their dataset (e.g., smart-gap, kpic), offering flexibility between conservative and aggressive trimming strategies.

To safeguard codon integrity, a custom Python script (`babappa_stopcodon_masker.py`) checks that trimmed alignments remain in-frame and free of errors. This step is critical: even a single frameshift can render an alignment incompatible with codon-based evolutionary models and bias selection inference. Sequences that fail this check are flagged and reported, enabling users to adjust trimming parameters or exclude problematic sequences. Additionally, when trimming introduces spurious internal stop codons, the script masks them to maintain compatibility with downstream analyses.

2.2.4 Recombination Detection (babappa_clipgard only)

Recombination is a major confounding factor in codon-based analyses of positive selection, as recombination breakpoints can mimic or obscure true adaptive signals. To address this, babappa_clipgard integrates the Genetic Algorithm for Recombination Detection (GARD) as implemented in HyPhy[10] though `script0.5.sh`. After trimming and frame validation, alignments are screened for evidence of recombination breakpoints. GARD employs a model selection approach based on genetic algorithms to partition alignments into putative non-recombining segments, reporting both the number and location of detected breakpoints. Alignments with strong evidence of recombination are flagged, and partitioned datasets are optionally provided for downstream codeml analyses. Partitioning is performed by the Python script `split_recombination_blocks.py`, which is integrated into `script0.5.py`.This allows users to explicitly account for recombination or to exclude affected genes from subsequent tests, depending on their study goals.

2.2.5 Phylogenetic Reconstruction

For each codon alignment, BABAPPA infers a maximum-likelihood phylogeny using IQ-TREE2 [7] under an appropriate codon substitution model. By default, we perform 1,000 ultrafast bootstrap replicates [12] and approximate likelihood-ratio tests (aLRT) to assess branch support. This step is executed by `script1.sh`, which collects the resulting tree files.

2.2.6 Foreground Branch Designation

A custom Python script (`4GroundBranchGenerator.py`) processes each inferred tree. It iteratively marks each branch in turn as the “foreground” branch (by appending a #1 tag) for subsequent branch-site tests [2]. The output is a set of Newick tree files, each with a different branch annotated as foreground.

2.2.7 Codeml Analyses

The codeml models were then fitted to each gene and branch scenario. Control files (`codeml.ctl`) for site, branch, and branch-site models are generated dynamically from a template using embedded `sed` commands, eliminating manual edits. For each gene, `script2.sh` runs branch and branch-site models (alternative and null) in parallel across branches using GNU Parallel[8], while `script3.sh` and `script4.sh` run site models (including M0, M1a, M2a, M3, M7, M8) in parallel. A Python wrapper (`run_codeml.py`) ensures that the correct model versions (alternative vs. null) are run for branch-site comparisons. Throughout, all executions leverage multi-core processing to maximize speed.

2.2.8 Result Aggregation and Statistical Testing

After model fitting, custom scripts (`script5.sh`, `script6.sh`) parse codeml output files to extract log-likelihood (*lnL*) values and parameter counts for each model, writing per-gene summaries in CSV format. A final Python script (`lrt_bh_correction.py`) computes likelihood-ratio statistics ($2\Delta lnL$) between paired models, calculates *p*-values from χ^2 distributions, and applies the Benjamini–Hochberg false-discovery rate (FDR) correction [13] across tests. Raw and adjusted *p*-values, along with significance flags (FDR < 0.05), are output to summary Excel reports in accordance with `script7.sh` and `script8.sh`. All pipeline steps and custom scripts are documented and configurable; source code is available on GitHub.

2.3. Benchmarking and statistical analysis

BABAPPA was compared to a conventional codeml workflow in which sequence alignments, phylogenies, and codeml runs were performed sequentially with without any parallelization. Both pipelines were executed on a consumer laptop (Intel Core i7 13th Gen CPU, 16 GB RAM, 512 GB SSD) running Ubuntu Linux via Windows Subsystem for Linux. For each of the three orthogroups, both BABAPPA and the standard workflow were run three times to account for run-to-run variability. Wall-clock times were recorded for each complete analysis (in seconds). The distributions of runtimes were compared using a two-sided Wilcoxon rank-sum test (Mann–Whitney U, $\alpha = 0.05$) to assess the statistical significance of speed differences. All tests and data analyses were conducted in R.

3. Results

3.1. Concordance of positive-selection calls

BABAPPA's automation did not compromise the ability to detect positive selection. Across all three Brassicaceae orthogroups, the babappa_normal workflow identified exactly the same set of genes under significant branch-site selection (after FDR correction, threshold < 0.05) as the conventional manual codeml workflow. Every gene flagged for positive selection by BABAPPA was also flagged by the manual approach, resulting in perfect concordance (Supplementary Files S2–S7). This demonstrates that automation of the codeml pipeline preserves the sensitivity and specificity of likelihood-ratio tests despite the removal of manual configuration steps.

The extended workflows further supported this finding. For babappa_clip, alignment trimming with ClipKIT followed by codon integrity validation produced selection results that were fully concordant with babappa_normal, indicating that trimming did not introduce false positives or cause loss of genuine signals. Instead, trimming improved alignment consistency by eliminating poorly aligned regions while preserving codon structure. Similarly, babappa_clipgard produced positive-selection calls consistent with the other two modes in alignments where no recombination was detected. In alignments where GARD reported evidence of recombination, BABAPPA flagged affected genes and provided recombination-partitioned alignments for downstream analyses. Although no recombination events were detected in the three Brassicaceae test orthogroups, these features provide safeguards for broader applications where recombination is more prevalent.

Together, these results confirm that BABAPPA's automation and optional extensions do not alter the underlying statistical inferences of codeml, while offering additional robustness against alignment artifacts and recombination.

3.2. Computational efficiency

In contrast to the manual workflow, BABAPPA dramatically reduced computation time. The mean wall-clock time across the three orthogroups was 1,430.2 seconds for babappa_normal, compared to 2,578.8 seconds for the standard sequential workflow. This represents a 44.6% reduction in runtime and an 80.3% increase in processing speed relative to the baseline. A Wilcoxon rank-sum test on the recorded runtimes yielded $p = 8.227 \times 10^{\square\square}$, indicating that

the efficiency gains are highly significant. Figure 2 illustrates the marked reduction in runtime achieved by BABAPPA.

The optional workflow extensions through GARD introduced modest additional costs. On average, incorporation of ClipKIT trimming and codon frame validation in babappa_clip does not increased runtime relative to babappa_normal, reflecting the added alignment filtering step does not consume significant time. But, the recombination analysis in babappa_clipgard further increased runtime by ~12–15%, depending on dataset size and GARD convergence speed. Nevertheless, even the most elaborate flavor of BABAPPA remained significantly faster than the conventional workflow, owing to extensive parallelization of codeml model fits and streamlined file handling.

Thus, BABAPPA achieves substantial efficiency gains while giving users the flexibility to trade small runtime increases for greater alignment robustness and recombination awareness (via ClipKIT and GARD).

4. Discussion

BABAPPA addresses long-standing technical and logistical bottlenecks in codeml-based selection analysis by unifying all key steps—quality control, codon-aware alignment, phylogenetic reconstruction, model fitting, and statistical reporting—into a single automated workflow. The pipeline eliminates the need for manual configuration of control files, iterative model execution, and error-prone collation of likelihood-ratio tests. Instead, it delivers an end-to-end solution that is fast, reproducible, and extensible. A major strength of BABAPPA is that it does not sacrifice accuracy for automation. The validation demonstrates that BABAPPA reproduces, without deviation, the same set of significant positive-selection calls as the traditional manual workflow. This concordance confirms that the gains in efficiency and reproducibility do not come at the expense of biological inference. Importantly, BABAPPA's modular design ensures that these inferences remain transparent: each intermediate file is preserved, and every filtering and statistical step is explicitly scripted. The three flavors of BABAPPA provide additional robustness. The babappa_normal mode implements the core workflow and already achieves substantial speedups through parallelization and optimized file handling. babappa_clip adds phylogeny-aware alignment trimming (ClipKIT) and codon-frame validation, reducing noise from misaligned or ambiguous regions while maintaining codon integrity. This extension is particularly useful

when dealing with divergent or unevenly aligned datasets. babappa_clipgard goes a step further by integrating recombination detection (via GARD), flagging or partitioning alignments where recombination may confound codeml inference. This safeguard is crucial for broader applications, such as viral or microbial datasets where recombination is common. These flavors allow researchers to choose between maximal speed and enhanced robustness, depending on their dataset and research question. It is important to note that BABAPPA cannot be directly benchmarked against existing selection-analysis pipelines, as none of the currently available tools integrate episodic selection tests (branch-site models) in the same systematic and automated fashion. Tools such as HyPhy offer site- and branch-based analyses, but their implementations and model frameworks differ fundamentally from codeml, making results non-equivalent. BABAPPA is therefore the first pipeline to offer a codeml-centered, fully automated framework for episodic selection detection across multiple genes and branches with integrated quality control, alignment curation, recombination safeguards, and statistical correction. Benchmarking against manual codeml workflows is thus the only meaningful point of comparison—and our results demonstrate that BABAPPA performs identically in terms of inference while delivering major gains in efficiency and reproducibility. Finally, BABAPPA's open-source and modular implementation ensures long-term adaptability. Users can adjust quality filters, substitute alternative aligners or phylogenetic inference tools, and even expand the pipeline to include additional evolutionary models. A graphical interface is under active development to lower the entry barrier for researchers less experienced with command-line workflows. As genomic datasets continue to grow, pipelines like BABAPPA will be essential for enabling rigorous, large-scale evolutionary studies in both high-resource and "dry-lab" contexts. In summary, BABAPPA provides the evolutionary biology community with a unique, efficient, and reproducible platform for positive-selection analysis. By integrating codon-aware quality control, phylogeny-aware alignment trimming, recombination detection, and episodic selection tests into a single workflow, it fills a critical methodological gap that no existing tool currently addresses.

5. Conclusion

The detection of positive selection is central to understanding molecular adaptation, yet conventional codeml workflows remain slow, error-prone, and difficult to scale. It also has very steep learning curve. BABAPPA addresses these limitations by automating codon-aware

filtering, alignment, phylogenetic inference, codeml execution, and FDR-corrected statistical testing in a fully reproducible framework. Validation on Brassicaceae orthogroups showed that BABAPPA reproduced all positive-selection calls of the standard manual workflow while reducing runtime by ~80%. Unlike existing pipelines, BABAPPA uniquely integrates episodic selection analyses, making direct benchmarking against other tools inappropriate; instead, performance is demonstrated against the conventional codeml standard. Its modular design, open-source availability, and new variants (babappa_normal, babappa_clip, and babappa_clipgard) make it adaptable to diverse research needs. By combining accuracy, efficiency, and reproducibility, BABAPPA lowers technical barriers and enables large-scale positive-selection analyses in evolutionary biology.

6. Funding details

This research received no funding.

7. Conflicts of interest

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

8. Data availability statement

The data are provided within the manuscript or supplementary information files. Also, the GitHub repository at https://github.com/sinhakrishnendu/babappa.git contains all the supplementary materials in the form of raw and processed data, as well as all necessary scripts to replicate the analyses presented in this manuscript.

9. Acknowledgments

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 was his creativity that gave the pipeline its name. I also thank Avijit Roy (Department of Zoology, Jhargram Raj College) for assistance in collecting the ortholog sequence data. I also acknowledge the developers of the open-source tools and databases used by BABAPPA, whose work made this pipeline possible.

10. Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the I used ChatGPT in order to refine this manuscript's and code's comments and readability. After using this tool/service, the I reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

11. References

1. Sinha, K., Jana, S., Pramanik, P. & Bera, B. Selection on synonymous codon usage in soybean (Glycine max) WRKY genes. *Scientific Reports 2024 14:1* **14**, 1–18 (2024).

2. Sinha, K. 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 (2025) doi:10.1101/2025.04.03.646976.

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

4. Álvarez-Carretero, S., Kapli, P. & Yang, Z. Beginner's Guide on the Use of PAML to Detect Positive Selection. *Mol Biol Evol* **40**, (2023).

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

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

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

8. Tange, O. GNU Parallel 2018. doi:10.5281/ZENODO.1146014.

9. Steenwyk, J. L., Buida, T. J., Li, Y., Shen, X. X. & Rokas, A. ClipKIT: A multiple sequence alignment trimming software for accurate phylogenomic inference. *PLoS Biol* **18**, e3001007 (2020).

10. Kosakovsky Pond, S. L., Posada, D., Gravenor, M. B., Woelk, C. H. & Frost, S. D. W. GARD: a genetic algorithm for recombination detection. *Bioinformatics* **22**, 3096–3098 (2006).

11. Bae, M., Yi, H. G., Jang, J. & Cho, D. W. Microphysiological Systems for Neurodegenerative Diseases in Central Nervous System. *Micromachines (Basel)* **11**, 855 (2020).

12. Hoang, D. T., Chernomor, O., Von Haeseler, A., Minh, B. Q. & Vinh, L. S. UFBoot2: Improving the Ultrafast Bootstrap Approximation. *Mol Biol Evol* **35**, 518–522 (2018).

13. Benjamini, Y. & Hochberg, Y. 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 (1995).

12. Tables

12.1. Table1

Details of scripts and their function with input and output format.

Script / File	Function	Input	Output
Babappa_normal.sh (in babappa-normal/)	Master driver for the classic BABAPPA workflow (babappa_normal). Orchestrates end-to-end QC → PRANK → IQ-TREE → codeml → aggregation.	input/ FASTA files (CDS)	Structured folders: QCseq/, msa/, treefiles/, codemloutput/, codemlanalysis/, BHanalysis/, Excel summary reports
babappa_clip.sh (in babappa-clip/)	Master driver for ClipKIT-augmented workflow (babappa_clip). Runs the normal pipeline but invokes ClipKIT rectification and stop-codon masking where configured.	input/ FASTA files (CDS)	Same structured folders as above; additionally clipmsa/ (or modified msa/) and frame/clip logs
babappa_clipgard.sh (in babappa-clipgard/)	Master driver for ClipKIT + GARD workflow (babappa_clipgard). Orchestrates trimming, frame checks, GARD recombination detection/partitioning, then downstream analyses.	input/ FASTA files (CDS)	Same structured folders plus clipmsa/, gard_results/, blocks/ (partitioned alignments), and recombination logs
seqQC.py	Performs biological QC on coding sequences (start/stop codons, length, outliers, allowed bases).	Raw FASTA (.fasta) or text (.txt) of coding sequences	QC-passed FASTA files in QCseq/; QC log file(s)
babappa_stopcodon_masker.py	Built-in stop-codon masker / frame-fix utility to ensure codeml compatibility after trimming/partitioning. Masks or removes problematic codons for downstream codon-model input.	Aligned FASTA (PRANK or ClipKIT output)	Masked/cleaned aligned FASTA; masker log
script0.sh … script8.sh (present in each flavour folder)	Modular scripts that implement the stepwise pipeline logic (QC, alignment, tree inference, model runs, parsing, aggregation). Each flavour provides these scripts with flavour-specific options (e.g., ClipKIT/GARD hooks).	QC-filtered FASTA / MSA inputs and templates	Intermediate and final outputs consistent with the pipeline (MSAs, trees, codeml outputs, CSVs)
run_codeml.py	Generates codeml control files from templates (site, branch, branch-site) and launches codeml runs (called by scripts).	Folder with aligned.fasta and Newick treefile	site/, branch/, branch-site/, branch-site-null/ result folders with codeml outputs and codeml.ctl
4GroundBranchGenerator.py	Generates per-tree Newick files where a single branch is annotated as foreground (#1) for branch-site tests.	One Newick treefile (.treefile)	Per-gene Newick files with foreground tags (placed in foregroundbranch/)
filter_blocks.py (in babappa-clipgard/ and babappa-gard/)	Post-GARD filter: selects/filters partitioned blocks from GARD output based on criteria (e.g., block size, support).	GARD output (JSON/CSV) and partitioned alignments	Filtered block lists; filtered block FASTA files
split_recombination_blocks.py (in babappa-clipgard)	Splits alignments into recombination-free blocks according to GARD breakpoints for independent downstream analysis.	GARD breakpoint files and original alignment	Block-specific FASTA files (one FASTA per block)
glue.sh (in babappa-clipgard/)	Glue/patch helper to reassemble or coordinate block-level results (used after block-wise analysis).	Block-level result folders	Reassembled/merged results or metadata used for downstream aggregation
script1.sh	Runs IQ-TREE2 codon phylogenetic inference (model selection, ultrafast bootstrap, aLRT) and collects tree files.	msa/*.fas or clipmsa/*.fas	iqtreeoutput/<gene>/ logs & treefiles; consolidated treefiles/
script2.sh	Parallelizes branch & branch-site codeml runs across species & treefiles.	Aligned FASTA (possibly block-split) & foregroundbranch/ treefiles; template codeml.ctl	codemloutput/<species>/<tree>_{B,BS,BS_NULL}/ with codeml outputs
script3.sh	Parallelizes M0 (site-null) codeml runs across species.	Aligned FASTA & treefiles; template codeml.ctl	codemloutput/<species>/M0/results with codeml outputs
script4.sh	Automates codeml site-model (M0, M1a, M2a, M3, M7, M8) analyses in parallel.	Aligned FASTA & treefiles; template codeml.ctl	sitemodel/<species>/ with model-specific outputs
script5.sh	Parses branch & branch-site codeml outputs to	codemloutput/<	codemlanalysis/<species

	extract lnL & np into CSV.	species>/.../output.txt files	>/lnL_np_values.csv
script6.sh	Parses site-model codeml outputs to extract lnL & np into CSV.	sitemodel/<species>/output.txt	sitemodelanalysis/<species>/lnL_np_values.csv
lrt_bh_correction.py	Computes LRT statistics and applies Benjamini–Hochberg FDR correction for branch/branch-site comparisons.	CSVs: codemlanalysis/<species>/*.csv	Excel reports BHanalysis/<species>/LRT_results_<…>.xlsx (raw p, FDR p, significance)
lrt_bh_correction.sitemodel.py	Computes LRT & BH correction for site-model comparisons and summarizes results.	CSV: site-model results (*.csv with Model, lnL, np)	SiteModelBH/<species>/lrt_results.csv (LRT, df, p, BH-adjusted p)
merge_bh_results.py	Merges BH-corrected results from multiple species/flavours into a consolidated report (helper script noted in README).	Multiple BHanalysis/ Excel/CSV outputs	Consolidated summary report (CSV/XLSX)

13. Figure legends

13.1. Figure 1

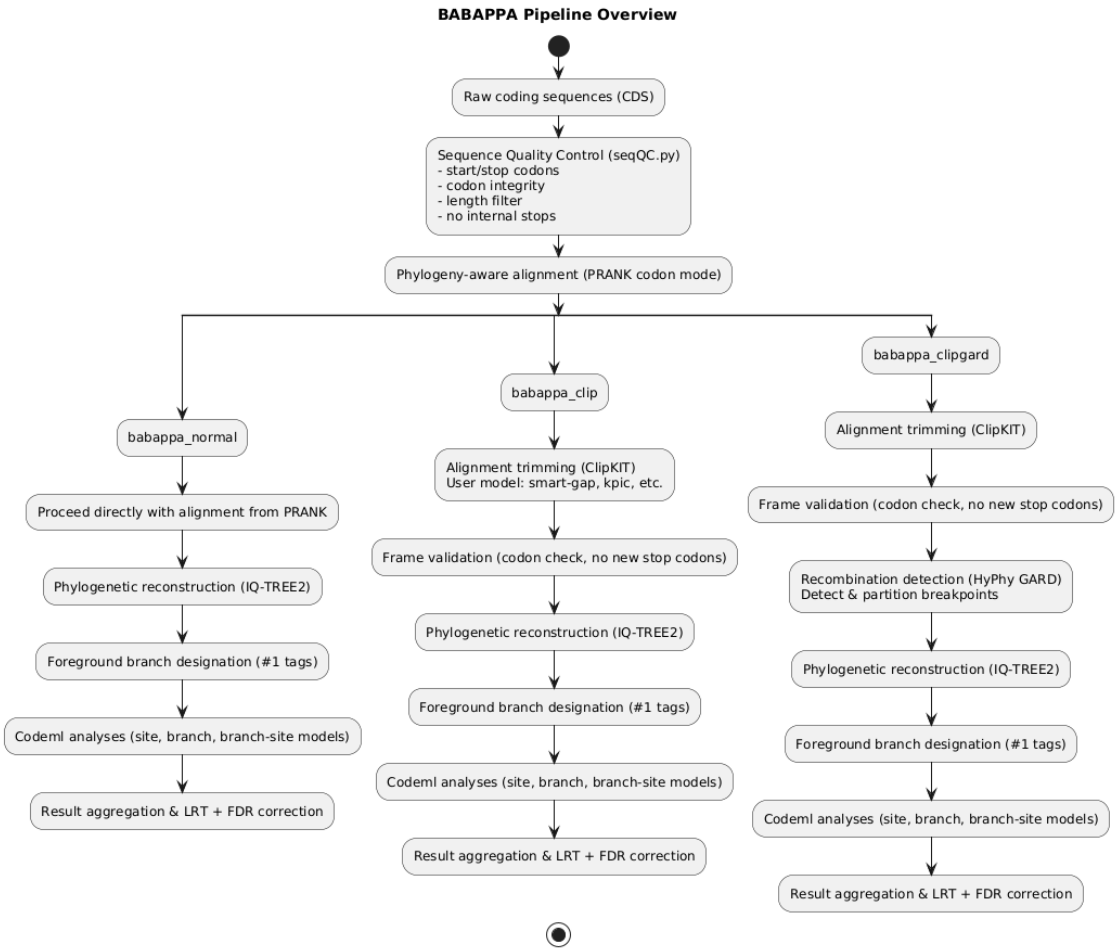
Schematic workflow of BABAPPA and comparative runtime performance against the standard codeml workflow.

13.2. Figure 2

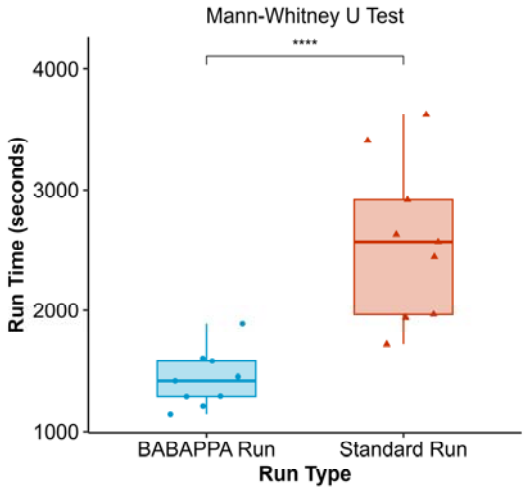
Benchmarking of BABAPPA run with standard run.

14. Figures

14.1. Figure 1


BABAPPA Pipeline Overview
Raw coding sequences (CDS)
Sequence Quality Control (seqQC.py)
- start/stop codons
- codon integrity
- length filter
- no internal stops
Phylogeny-aware alignment (PRANK codon mode)
babappa_normal
Proceed directly with alignment from PRANK
Phylogenetic reconstruction (IQ-TREE2)
Foreground branch designation (#1 tags)
Codeml analyses (site, branch, branch-site models)
Result aggregation & LRT + FDR correction
babappa_clip
Alignment trimming (ClipKIT)
User model: smart-gap, kpic, etc.
Frame validation (codon check, no new stop codons)
Phylogenetic reconstruction (IQ-TREE2)
Foreground branch designation (#1 tags)
Codeml analyses (site, branch, branch-site models)
Result aggregation & LRT + FDR correction
babappa_clipgard
Alignment trimming (ClipKIT)
Frame validation (codon check, no new stop codons)
Recombination detection (HyPhy GARD)
Detect & partition breakpoints
Phylogenetic reconstruction (IQ-TREE2)
Foreground branch designation (#1 tags)
Codeml analyses (site, branch, branch-site models)
Result aggregation & LRT + FDR correction

14.2. Figure 2


Mann-Whitney U Test
****
4000
3000
2000
1000
Run Time (seconds)
BABAPPA Run
Standard Run
Run Type