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:

Detecting signals of positive selection using codon substitution models requires complex workflows that integrate sequence alignment, recombination screening, tree inference, and multiple codeml runs. Existing command-line pipelines demand considerable technical expertise and can be error-prone. Here BABAPPA (Bash-Based Automated Parallel Positive Selection Analysis) has been introduced, a Codeml-centered automated pipeline with an integrated graphical user interface (GUI) designed to make advanced positive selection analysis accessible to both computational and experimental biologists. BABAPPA implements three complementary workflows (normal, clip, and clipgard) that automate alignment curation, optional recombination-aware partitioning, tree reconstruction, codeml analyses, and multiple-testing correction. The GUI simplifies execution, manages background processes inside WSL (Ubuntu-22.04), and presents outputs in an organized, publication-ready structure. Benchmarking on Brassicaceae orthogroups confirmed that BABAPPA reproduced manual codeml results while reducing runtime from 2,579 seconds to 1,430 seconds, a 44.6% reduction (1.8 times faster). By combining automation, reproducibility, and an intuitive GUI, BABAPPA lowers the barrier to performing rigorous positive selection analysis and enables wider adoption of codeml-based evolutionary inference.

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

1. Introduction

The detection of positive (adaptive) selection in protein-coding genes is a cornerstone of molecular evolution, providing insights into adaptive changes across lineages and taxa [1,2]. Among the available approaches, maximum-likelihood models implemented in codeml within the PAML package, including site, branch, and branch-site models, have long been regarded as the gold standard for identifying such signals [3]. Despite their reliability, traditional codeml workflows are labor-intensive. Researchers must manually edit control files, run multiple model comparisons sequentially, collate likelihood-ratio test (LRT) results across many genes and branches, and apply multiple-testing corrections [4]. These steps are prone to human error and pose a significant barrier to throughput when analyzing large gene families or genome-scale datasets. To overcome these challenges, BABAPPA (Bash-Based Automated Parallel Positive Selection Analysis) has been developed. It's an end-to-end pipeline that automates all stages of codeml analysis, from raw sequence input to statistically validated results. BABAPPA streamlines sequence quality control, codon-aware alignment, phylogenetic inference, branch annotation, model fitting, and LRT-based testing, while exploiting parallel execution to maximize computational efficiency. It combines widely used bioinformatics tools, including Biopython [5] for sequence handling, PRANK [6] for codon alignment, IQ-TREE2 [7] for phylogenetic inference, and GNU Parallel [8] for workload distribution, with custom scripting to deliver a seamless and reproducible workflow.

BABAPPA is available in two complementary modes. The command-line implementation offers maximum flexibility, allowing advanced users to fine-tune every stage of the workflow. In parallel, the graphical user interface (GUI) lowers the barrier for non-specialists by managing the backend automatically while providing user-friendly options alongside sensible presets that reduce complexity. This dual design ensures that BABAPPA serves both computational biologists seeking customization and experimental researchers who require an accessible, turnkey solution. Within both interfaces, BABAPPA supports three pipeline flavours. The babappa_normal workflow performs codon alignment, tree reconstruction, and codeml-based detection of selection. The babappa_clip workflow extends the normal mode by incorporating ClipKIT [9] for alignment trimming, offering user-selectable strategies (e.g., smart-gap or kpic in the command-line version) and applying built-in quality control to preserve codon frames and flag premature stop codons. The babappa_clipgard workflow further integrates HyPhy GARD [10] to detect recombination breakpoints, thereby

accounting for the confounding effects of recombination on positive selection inference. Unless otherwise specified, smart-gap is used as the default ClipKIT strategy, such in GUI.

Together, these features enhance robustness, flexibility, and reproducibility while minimizing manual intervention. This study describes the design of BABAPPA, demonstrates its runtime efficiency compared to conventional manual codeml workflow, and highlights its reliability in reproducing manual results. Unlike existing software tools that typically address only individual workflow stages (e.g., alignment, tree inference, or recombination detection), BABAPPA provides a unified, reproducible, and scalable codeml-based framework. Rather, than competing with such tools, BABAPPA integrates them into a single automated pipeline available through both CLI and GUI, offering the first practical end-to-end solution for large-scale positive selection analysis.

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 Table 1, enabling independent replication of the 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`) (Table 2). 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 normal, these alignments

proceed directly to phylogenetic inference, whereas in clip and clipgard, they undergo additional refinement through trimming and validation (see below).

2.2.3 Alignment Trimming and Frame Validation (clip and clipgard only)

Alignment trimming removes ambiguous or poorly supported regions from multiple sequence alignments. In the clip and 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. Graphical User Interface (GUI)

BABAPPA is distributed not only as a Bash-based command-line pipeline but also as a GUI designed to make positive selection analysis accessible to researchers with limited computational expertise. The GUI encapsulates the same automated workflows as the command-line version but presents them through an intuitive point-and-click interface, minimizing manual intervention.

The GUI runs on Windows and manages all backend operations transparently through the Windows Subsystem for Linux (WSL, Ubuntu-22.04). On first launch, it automatically installs or restores the required conda environment, deploys the BABAPPA project files into the WSL environment, and starts the FastAPI-based backend server. These steps are handled entirely in the background, enabling the user to begin analysis without interacting with the command line. Within the GUI, users select input FASTA files and designate output directories using file pickers (Figure 2). Pipeline selection is made through a simple dropdown menu that allows the choice between normal, clip, and clipgard workflows. In the clip and clipgard modes, alignment trimming is performed by ClipKIT using the smart-gap strategy by default. Smart-gap was chosen because it provides a robust, general-purpose trimming method: it adapts its threshold to the distribution of gaps within each alignment, avoids the over-trimming seen with stricter fixed thresholds, and consistently retains informative sites across a wide variety of datasets[14]. Benchmarking of ClipKIT has demonstrated that smart-gap balances site retention with accuracy, yielding reliable phylogenetic and downstream codeml inference[14]. For these reasons, smart-gap is the recommended default by the ClipKIT developers and has been adopted as the standard in BABAPPA's automated workflows[14]. When the clipgard pipeline is selected, the GUI extends the trimming workflow by automatically invoking HyPhy GARD to detect recombination breakpoints, thereby reducing the risk of false positive signals of selection caused by recombination. All available CPU cores are automatically detected and utilized to maximize efficiency, relieving users from the need to configure performance settings manually. The GUI also includes a live log panel that displays progress messages from the backend in real time, making it possible to follow each step of the workflow. A "Clear

Session" function is provided to reset the analysis state and clear temporary files, preparing the interface for subsequent runs without affecting saved results.

By encapsulating the complexity of the Bash-based workflow within a user-friendly graphical interface, BABAPPA enables codeml-based positive selection analyses to be launched and completed with just a few clicks. At the same time, the GUI preserves all intermediate outputs, logs, and results, ensuring reproducibility and transparency. Together with the command-line implementation, the GUI broadens the accessibility of BABAPPA, making it suitable for both computational specialists requiring flexibility and experimental researchers seeking an accessible, reproducible solution.

2.4. 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 preserved the accuracy of codeml's positive-selection detection. Across all Brassicaceae orthogroups, normal flavor reproduced exactly the same results as the manual codeml workflow (FDR < 0.05), demonstrating perfect concordance. The extended flavors supported this finding: clip improved alignment consistency through ClipKIT trimming, while clipgard matched clip mode when no recombination was detected and flagged genes with evidence of recombination for partitioned analysis. Thus, BABAPPA maintains codeml's sensitivity and specificity while reducing artifacts from misalignment and recombination and significantly lowers computational time.

3.2. Computational efficiency

BABAPPA has been evaluated using three Brassicaceae orthogroups (five sequences each) (table 1) to compare runtime performance and accuracy with standard manual codeml workflows. Across all runs, BABAPPA produced identical likelihood values, parameter estimates, and LRT outcomes to the manual analyses, confirming methodological reproducibility. In terms of efficiency, the total runtime decreased from 2,579 seconds (manual) to 1,430 seconds (BABAPPA), corresponding to a 44.6% reduction in execution time or an approximate 1.8× speedup. This demonstrates that BABAPPA not only reproduces codeml results faithfully but also substantially accelerates the analysis by leveraging automated parallelization and streamlined workflow management.

Importantly, the GUI successfully managed backend execution inside WSL, handled environment setup automatically, and organized results into a standardized folder hierarchy. This confirms that the GUI is not only functional but also lowers the technical barrier for non-specialist users, allowing analyses to be launched with a few clicks while still producing publication-ready outputs.

4. Discussion

Codon-based models of molecular evolution, particularly those implemented in codeml, remain essential for detecting positive selection at both site and branch levels. However, the complexity of preparing alignments, partitioning datasets, selecting tree models, and interpreting multiple codeml outputs often restricts these analyses to technically proficient users. Several command-line pipelines exist to streamline specific steps, but they still demand substantial manual configuration and offer little support for less experienced researchers. BABAPPA addresses these challenges by combining a fully automated backend with an intuitive GUI. By embedding workflow automation in a GUI that runs seamlessly on Windows through WSL, BABAPPA lowers the technical barrier for evolutionary biologists who may lack extensive computational expertise. The GUI not only manages backend processes and environment setup automatically but also standardizes outputs into a consistent structure, making downstream interpretation and publication preparation more straightforward. Performance benchmarking demonstrated that BABAPPA achieves identical analytical results to manual codeml workflows while delivering a ~45% reduction in runtime (1.8× faster) through parallelization and streamlined job handling. These gains highlight that

accessibility does not come at the expense of rigor or efficiency. Beyond runtime improvements, BABAPPA also promotes reproducibility and transparency by packaging its environment, providing clearly organized logs, and reducing the risk of human error in repetitive codeml runs. This combination of automation and accessibility ensures that advanced evolutionary analyses can be performed consistently across research groups with varying computational expertise.

BABAPPA shares some aims with tools such as EasyCodeML [15], AlexandrusPS [16], and GUIs like pamlX [17], which provide interactive or semi-automated access to codon-based selection analyses. However, unlike these, BABAPPA is the first pipeline to integrate selection models fully within an end-to-end automated framework, incorporating alignment trimming (ClipKIT), recombination detection (HyPhy GARD), sequence quality control, phylogenetic inference, codeml model fitting, and FDR-corrected statistical testing. Direct benchmarking against these existing tools is not practicable, because many do not support exactly the same set of models, do not handle recombination or alignment trimming systematically, or require substantial manual parameter tuning (e.g., labeling tree branches, control file setup) that BABAPPA automates. For example, EasyCodeML offers GUI presets and custom modes but does not systematically incorporate recombination detection or modular flavours with automatic trimming. What makes BABAPPA unique is its modular design with three flavours (normal, clip, clipgard) that let users choose between speed, alignment robustness, or recombination awareness; preservation of all intermediate files and fully scripted steps for transparency; and, importantly, the addition of a GUI for users without command-line experience. Looking ahead, the modular structure of BABAPPA allows for future expansion, such as integration of additional alignment tools, alternative phylogenetic inference methods, or visualization modules directly into the GUI. By bridging the gap between powerful evolutionary models and user-friendly execution, BABAPPA represents a step toward more inclusive and reproducible computational biology.

5. Conclusion

BABAPPA offers a dual solution for codeml-based positive selection analysis: a flexible Bash-based command-line pipeline for advanced users who require fine-grained control, and an intuitive GUI for researchers who prefer simplicity and guided execution. The command-line implementation preserves full transparency and flexibility, while the GUI provides features such as model selection for ClipKIT, adjustable levels of parallelism, and carefully

chosen presets that reduce complexity for non-specialist users. Together, these two complementary modes ensure that BABAPPA can serve both computational biologists seeking custom workflows and experimental researchers who need accessible, reproducible analyses with minimal setup. By combining automation, efficiency, reproducibility, and usability, BABAPPA represents a novel and practical framework for large-scale positive selection studies, lowering barriers to entry while maintaining scientific rigor.

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/krishnendusinha/BABAPPA. 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.17211886. 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

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).

Group	Species	Gene IDs
Group 1	***Arabidopsis helleri***	Araha.2210s0007.1; Araha.22349s0004.1; Araha.23363s0006.1; Araha.23705s0002.1; Araha.2816s0002.1
	Arabidopsis lyrate	AT1G68520.1; XP_020883914.1; XP_002868216.1; XP_020868537.1; XP_002893503.1
	Arabidopsis thaliana	AT1G25440.1; AT1G28050.1; AT1G49130.1; AT1G49130.2; AT1G68520.1
Group 2	***Arabis alpina***	KFK31035.1; KFK32611.1; KFK36040.1; KFK37540.1; KFK37702.1
	Capsella rubella	Carubv10009731m; Carubv10013971m; Carubv10015501m; Carubv10016442m; Carubv10020384m
	Eutrema salsugineum	Carubv10020384m; Thhalv10007673m; Thhalv10007685m; Thhalv10011633m; Thhalv10011649m
Group 3	***Arabidopsis thaliana***	AT1G49130.2; AT1G68520.1; AT1G73870.1; AT2G24790.1; AT2G33500.1
	Camelina sativa	Csa16g037740.1; Csa16g046500.1; Csa17g035050.1; Csa17g037850.1; Csa17g080770.1
	Capsella grandiflora	Cagra.0876s0020.1.p; Cagra.1004s0003.1.p; Cagra.11928s0002.1.p; Cagra.1194s0065.1.p; Cagra.14026s0004.1.p

14. Advanced Usage — clipkit documentation. at <https://jlsteenwyk.com/ClipKIT/advanced/index.html?utm_source=chatgpt.com>

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

16. Ceron-Noriega, A., Schoonenberg, V. A. C., Butter, F. & Levin, M. AlexandrusPS: A User-Friendly Pipeline for the Automated Detection of Orthologous Gene Clusters and Subsequent Positive Selection Analysis. *Genome Biol Evol* **15,** (2023).

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

8. Tables

8.1. Table 1

Details of ortholog gene groups used in the study.

8.2. Table 2

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)
script1.5.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 extract lnL & np into CSV.	codemloutput/<species>/.../output.txt files	codemlanalysis/<species>/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)

9. Figure legends

9.1. Figure 1

Schematic workflow of BABAPPA.

9.2. Figure 2

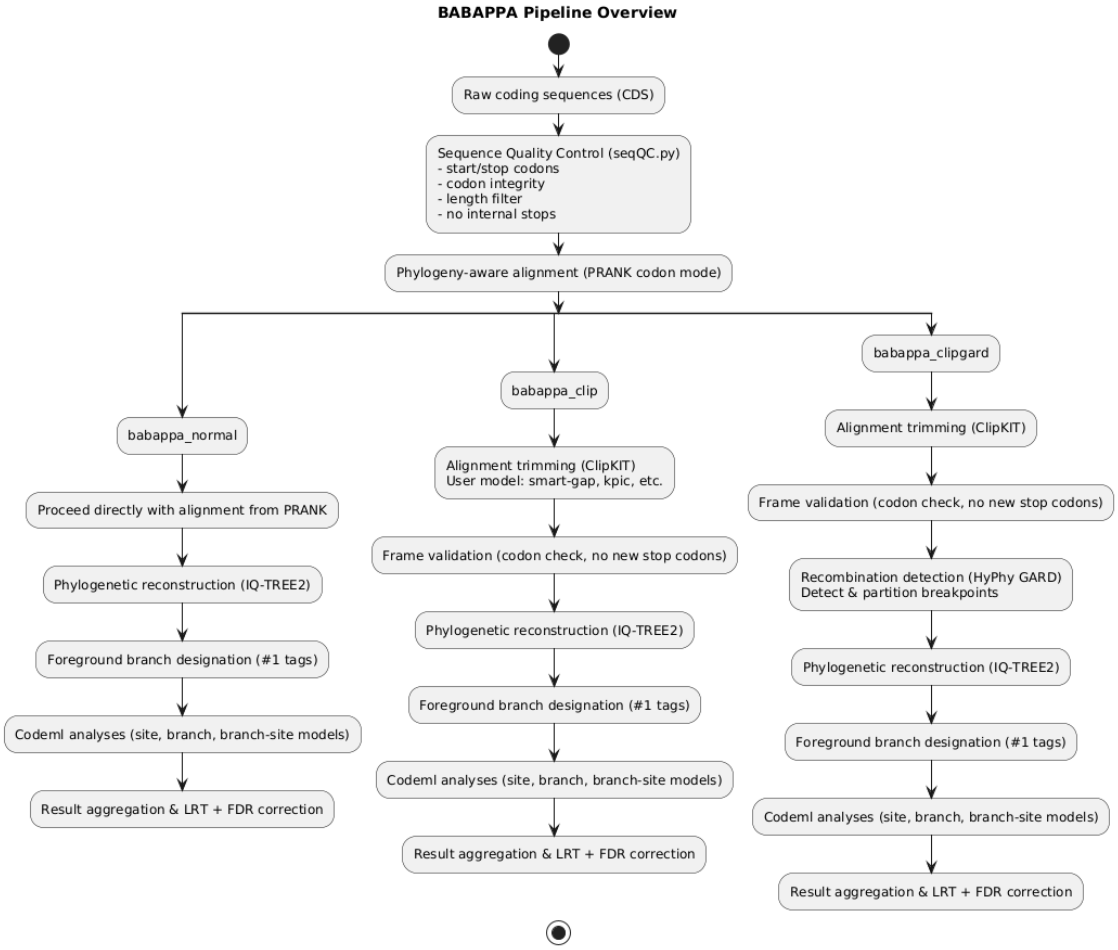
BABAPPA Graphical User Interface.

9.3. Figure 3

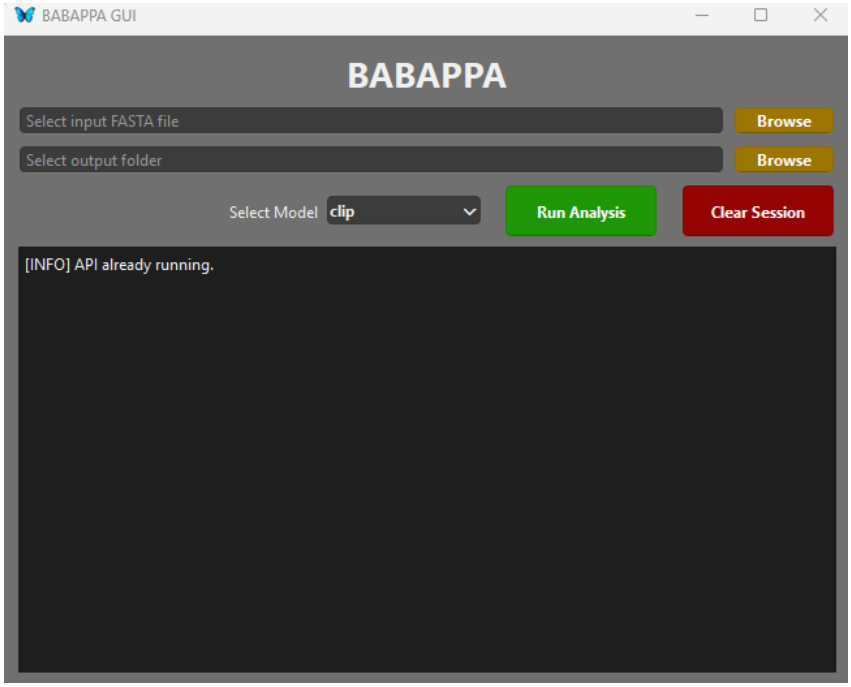
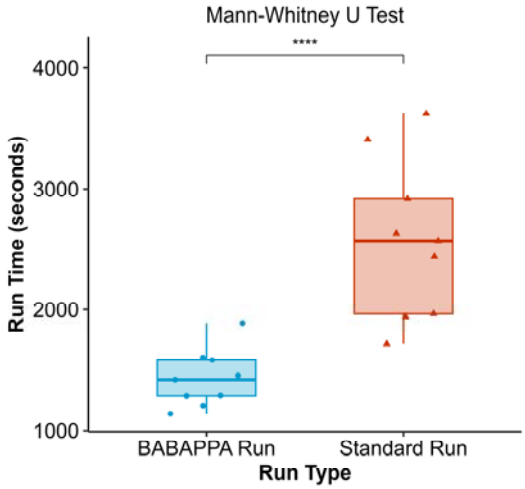
Benchmarking of BABAPPA run with standard run.

10. Figures

10.1. Figure 1



10.2. Figure 2



10.3. Figure 3