Installation Recommendations

In general, CASA requires internet connection, Python 3.7+ (with pandas and requests libraries installed), and installations of Clustal Omega (http://www.clustal.org/omega/) and NCBI BLAST+ (https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/). Additionally, ~1 GB of extra storage is recommended for Swiss-Prot downloads and BLAST databases. Any workflow that requires Clustal Omega and BLAST must ensure that the parent directories of the clustalo and blastp executables are added to PATH; the Clustal Omega executable, specifically, may need to be renamed "clustalo". In all workflows, the directory for CASA itself should be added to PYTHONPATH. The latest version of CASA was built and tested on Ubuntu 24.04.1 LTS with Python 3.12.3 (pandas 2.2.3 and requests 2.32.3), Clustal Omega 1.2.4 (Sievers et al., 2011; Sievers and Higgins, 2018), and BLAST 2.16.0+ (Camacho et al., 2009); different environments and dependencies may have different options and behaviors.

Protocols

Commands are demonstrated on Ubuntu 18.04.5 LTS. They are indicated by "$", but the "$" should not be copied into the terminal.

Basic Protocol: *Organization and visualization of target proteins*

Visualization of functional regions within proteins is often needed prior to experimental characterization. This is usually done by comparing target proteins to well-annotated references, and can, itself, be a tedious process. The basic protocol is meant to simplify and organize the

generation of files used in detailed protein annotation workflows, and is recommended as a first look at a new sequence.

Requirements:

- Internet connection
- Python 3.7+ (with pandas and requests libraries installed)
- ~1 GB of storage (if Swiss-Prot database is to be downloaded/created)
- Clustal Omega (Sievers et al., 2011; Sievers and Higgins, 2018) (http://www.clustal.org/omega/), installed as clustalo
- blastp, from NCBI BLAST+ (Camacho et al., 2009) (https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/)

Workflow summary: The basic protocol takes one FASTA file with at least one target protein as input. It runs all four scripts in CASA.py on each target protein from the input FASTA file. The main results of each script on one target protein are as follows:

- search_proteins.py
    - blastp is run with a target protein query against a Swiss-Prot subject database; results are saved to a table (query.tsv).
    - Top BLAST hit sequences are downloaded through UniProt's REST API (The UniProt Consortium, 2024) and combined into a single FASTA file along with the target protein (all.fasta).
- retrieve_annotations.py
    - Annotations for each BLAST hit are retrieved through UniProt's REST API (The UniProt Consortium, 2024) and saved to a table (all.ann).
- alignment.py

    - The all.fasta output from search_proteins.py is aligned using a call to clustalo, resulting in a Clustal alignment (alignment.clustal).
- clustal_to_svg.py
    - all.ann and alignment.clustal, from retrieve_annotations.py and alignment.py, respectively, are used to generate an annotated set of SVGs (0.svg, 1.svg, etc.).

The following options were specified in this example:

- -nr 5 OR --num_res 5: The --num_res option sets the intended number of Swiss-Prot subject hits. It is equivalent to setting -num_alignments 5 in a blastp run with the same inputs. Despite the relative depth of Swiss-Prot annotations, not all proteins are annotated equally. It may also take a combination of proteins to appropriately annotate a novel target protein with unusual domains. That said, hits that are too divergent can lead to poor Clustal alignments. We therefore recommend starting with five BLAST hits in order to balance depth of reference annotations with quality of the resulting alignment.
- -n FALSE OR --nums FALSE: The --nums option adds numbers back into SVG alignments, in the manner of .clustal_num or .aln-clustal_num files. They can be useful for identifying residue numbers, but the protein names are offset. We do not recommend using them for final figures.
- -c FALSE OR --codes FALSE: The --codes option adds Clustal conservation codes (“.”, “:”, “*”) back into SVG alignments. They can be useful for identifying patterns in conservation of biochemical properties at each position. Like --nums, we do not recommend using them for final figures, as they tend to clutter the image.
- -u TRUE OR --uniprot_format: The --uniprot_format option truncates protein names as if they were UniProt entries. For example, “sp|Q766C3|NEP1_NEPGR” becomes

"NEP1_NEPGR". This ensures that protein names will be truncated in a way that still retains information, rather than being cut short due to page width limitations. It should not affect most non-UniProt names, but may have unintended results if non-UniProt names include "|". We generally recommend using --uniprot_format TRUE.

Steps and example commands:

1. **Run CASA from the command line.** The basic protocol consists of just one command, which should run from any command-line interface. As it runs, it will write a log of commands and key information to the command interface; for large runs, we recommend redirecting this output to a file for future reference. Notably, annotated feature residues and alignment positions will be written as SVG files are generated, making it possible to tell which alignment SVGs contain target annotations.

```
$ python -m CASA -i ./target_proteins.fasta -o ./outputs -ord blast annotate align svg -nr 5 -n FALSE -c FALSE -u TRUE
```

2. **Choose one target protein, navigate to the appropriate folder (./outputs/target_protein), and view alignment SVGs.** The chosen folder should contain a number of files that can be used for downstream analyses. Among them should be one or more SVG files that can be viewed in an SVG editor or browser. Note that text is only fully editable when opened in Inkscape; however, it is viewable in Adobe Illustrator, Google Chrome, and Microsoft Edge. Each SVG contains part of an annotated alignment; we would therefore recommend consulting the log output for the SVG locations of target features**.** For example, signal sequences are typically found at the N-terminus; if the chosen target protein contains a signal sequence, it is likely to be found in 0.svg.

3. **Identify putative features in the target protein by comparing its sequence with those of annotated references.** In this example, each target protein was aligned with up to five Swiss-Prot proteins. Generally, residues in novel proteins that align with known features in reference proteins will serve similar functions. A putative signal sequence in the target protein can be estimated by comparing the boundaries of annotated signal sequences in the aligned Swiss-Prot proteins.

Extra Step 1: ***Validate putative features in the target protein by employing other bioinformatics tools.***

The presence of a feature in reference proteins makes it more likely that the same feature exists in the target protein, particularly if the feature is strongly conserved among several reference sequences. Even so, it is sometimes difficult to tell exactly where feature boundaries lie. Signal sequences, for example, have a conserved, helical structure, but are not often conserved in sequence (Bruch et al., 2002; Yuan et al., 2003). It can therefore be difficult to identify signal sequence cleavage sites by visual alignment alone. Instead, tools like SignalP-6.0 (Teufel et al., 2021) use language models to confirm the presence of signal sequences and predict their cleavage sites. CASA can give the user a strong first hint for functional annotation, but it helps to have outside validation when dealing with novel or poorly-conserved sequences.

Extra Step 2: ***Use molecular modeling techniques to characterize interactions between annotated features.***

There are limitations to the information that can be inferred from primary sequence conservation alone. Structure and function annotations in particular can benefit from molecular modeling

techniques like protein structure prediction and molecular dynamics simulations. For example, though functions like binding and catalysis are often conserved between related proteins, it is not always the same residues that carry out those functions. Even when important residues themselves are conserved at the primary sequence level, their capacity for molecular interaction may vary depending on their orientation in space. It is not impossible for the function of conserved features to be carried out by nearby residues if their 3D orientations are unfavorable. Protein structure prediction tools like D-I-TASSER (Zheng et al., 2023, Anon, n.d.) can provide an initial snapshot of these orientations, while molecular dynamics simulations can show how they might change over time. Together, they show which molecular interactions are most plausible, and contribute to alternative hypotheses about functional residues.

Extra Step 3: ***Improve alignments by using alternative inputs.***

The basic protocol records up to five BLAST subject sequence hits via the --num_res option, but alignments from those hits may not always contain useful information. Lenient thresholds (blastp defaults) may identify hits that align poorly with the given query; this is more likely with novel or under-characterized families of proteins, where fewer well-annotated examples exist. Relatedly, Swiss-Prot is limited to proteins that have been manually-reviewed. There are proteins outside of the Swiss-Prot database that have useful annotations and may contain sequence motifs more similar to those of the target protein. For example, cleavage sites for propeptides are difficult to predict if cleavage site residues are not conserved among Swiss-Prot reference proteins. Non-Swiss-Prot references that share greater sequence similarity with the target protein can strengthen these cleavage site predictions. If these proteins are known, CASA can be re-run from the alignment or annotation step using a manually-curated FASTA file:

$ python -m CASA -i ./outputs/target_protein/modified_all.fasta -
o ./outputs/target_protein/modified -n FALSE -c FALSE -u TRUE -ord align annotate svg

Extra Step 4: ***Re-generate alignment SVGs with Clustal features.***

The output alignment from the basic protocol is intended to improve alignment readability. It is therefore missing some features when compared with a standard .clustal_num or .aln-clustal_num alignment: residue numbers next to each line, and conservation codes under each block ("*", ".", ":", according to https://ebi-biows.gitdocs.ebi.ac.uk/documentation/faqs/clustal/). It can be helpful to visualize alignments with these added features. For example, ":" and ".", which indicate strong and weak conservation respectively, can be used to identify biochemically conserved regions with non-identical residues. **Extra Step 2** can improve interpretability by adding detail back into previously simplified alignments.

Users can re-generate alignment SVGs generated in the basic protocol by running CASA with only the --order svg option, along with the appropriate inputs:

$ python -m CASA -i ./outputs/target_protein/alignment.clustal -o ./outputs -
a ./outputs/target_protein/all.ann -ord svg -n TRUE -c TRUE -u TRUE

Extra Step 5: ***Find residue numbers using initial IDs.***

There are two features that make finding residue numbers easier: the --nums option, and the initial IDs for each line and character. While the --nums option does allow users to visualize residue numbers, they must still count individual residues themselves. We instead recommend the built-in IDs for each residue.

When CASA first generates an SVG, it assigns meaningful IDs to most characters and lines; these IDs may be rewritten if the SVG is saved in an editor like Inkscape or Illustrator. Residues have unique IDs in the format: [PROTEIN]:[AA]#, where [PROTEIN] is the full protein name (before truncation due to UniProt formatting or size constraints), [AA] is the one-letter amino acid code, and # is the residue number in the unaligned sequence. In Inkscape, residue IDs can be found by using the XML editor; character-like elements, including residues, are nested within line-like elements, which are in turn contained within a text element that is grouped with a guide rectangle. Expanding the "Layer 1" group and the text element in the Inkscape XML editor reveals lines with ID format: [PROTEIN]#, where [PROTEIN] is the full protein name and # is the alignment line number. These lines can, in turn, be expanded to show individual residue IDs. Selecting an element (like a line or character) with the selector tool in the XML editor also selects it on the page, which allows the user to correlate residue numbers with positions in the alignment.

Extra Step 6: ***Re-generate alignment SVGs with custom annotations.***

Swiss-Prot entries are well annotated, but those annotations are often limited to widely-used and validated structural features. Novel or lesser-studied regions can still be useful in protein characterization. **Extra Step 6** addresses this need by allowing users to add custom feature annotations to their SVG alignments. It can be used to test near publication-ready annotations and to visualize regions within proteins that are not stored in UniProt. A common use case may be to visualize domains on a cluster of novel proteins with few known homologs.

Extra Step 6 makes use of the --features option (-f OR --features), which allows users to customize annotated features and their SVG colors. It should be passed as a comma separated list

of feature:color pairs, enclosed in quotes. Spaces should not be included before or after commas or colons, but they are permissible within feature names: “Active site:blue,Signal:red” is valid, but “Active site:blue, Signal:red” is not. --features is compatible with any colors valid for the “fill” attribute in SVG styling, including hexadecimal notation and color keywords. Feature matching is case-sensitive; features passed to --features must exactly match an included “type” in annotation files in order to be counted. Default feature:color combinations are listed in the https://github.com/grtakaha/CASA/ README.

Starting with outputs from the basic protocol, an example command that uses custom annotations might look like:

```
$ python -m CASA -i ./outputs/target_protein/alignment.clustal -
a ./outputs/target_protein/all_custom.ann -o ./outputs/target_protein/custom_annotations -u
TRUE -c FALSE -n FALSE -ord svg -f "Active site:#0000ff,Disulfide
bond:#e27441,Propeptide:#9e00f2,Signal:#2b7441,CUSTOM FEATURE:red"
```

References

Anon. D-I-TASSER: deep learning-based protein structure prediction. Website https://zhanggroup.org/D-I-TASSER/ [accessed 6 December 2024].

Bruch, M. D., C. J. McKnight, and L. M. Gierasch. 2002. Helix formation and stability in a signal sequence. *ACS Publications*. Website https://pubs.acs.org/doi/pdf/10.1021/bi00447a043 [accessed 6 December 2024].

Camacho, C., G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T. L. Madden. 2009. BLAST+: architecture and applications. *BMC bioinformatics* 10: 421.

Sievers, F., and D. G. Higgins. 2018. Clustal Omega for making accurate alignments of many protein sequences. *Protein Science* 27: 135–145.

Sievers, F., A. Wilm, D. Dineen, T. J. Gibson, K. Karplus, W. Li, R. Lopez, et al. 2011. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology* 7: 539.

Teufel, F., J. J. A. Armenteros, A. R. Johansen, M. H. Gíslason, S. I. Pihl, K. D. Tsirigos, O. Winther, et al. 2021. SignalP 6.0 achieves signal peptide prediction across all types using protein language models.

The UniProt Consortium. 2024. UniProt: the Universal Protein Knowledgebase in 2025. *Nucleic Acids Research*: gkae1010.

Yuan, Z., M. J. Davis, F. Zhang, and R. D. Teasdale. 2003. Computational differentiation of N-terminal signal peptides and transmembrane helices. *Biochemical and Biophysical Research Communications* 312: 1278–1283.

Zheng, W., Q. Wuyun, P. L. Freddolino, and Y. Zhang. 2023. Integrating deep learning, threading alignments, and a multi-MSA strategy for high-quality protein monomer and complex structure prediction in CASP15. *Proteins: Structure, Function, and Bioinformatics* 91: 1684–1703.