General:

- Spike data hierarchy: multiple physical *probes* -> multiple physical *channels* (electrodes) -> multiple informational/representational/calculated *units* (neurons)
- Longleaf settings:
    - GPU regular instead of FULL
    - A100
    - Additional job submission arguments:
    - --mem=64G (up to 118)
    - Jupyter
    - */proj/STOR/pipiras/Neuropixel*
- Have to train per subject (probably cross session)
- Zinodo macak cursor data (neurolink) for more data
- Environment:
    - Metadata shows pipeline using Python 3.9.7
    - Python 3 (ipykernel): Using python 3.11, torch 2.4.1+cu121 cuda: 12.1,
        - If using PyG need a PyG build that matches Torch 2.4.1/cu121
    - From "module avail", "python/3.9.6" is an option I can switch to, but should probably stick to newer kernel env bc backwards compatibility with old torch versions will probably cause bigger issues with newer CUDA driver.

Files Types:

- **.pkl** — Python pickles holding model-ready arrays/DataFrames.
- **.pt** — PyTorch tensors (not models) such as the VAE latent encodings of each stimulus image used for feature-space comparisons.
- **.pth** — PyTorch model checkpoints (parameters) for trained networks (e.g., Transformer→LSTM heads that predict spikes/firing).
- **.nwb** — Allen/Neurodata Without Borders session files with raw data and metadata. Contains full unprocessed session.
- **.csv** — Flat tables for metadata and small numeric summaries (e.g., units.csv, channels.csv, sessions.csv) used for joins/filtering.
- **.json** — Manifests/config/settings and SDK cache info (e.g., ecephys_project_manifest.json, session maps, preprocessing configs). Parameters for data fetching from SDK.

Boc: folder with unused .nwb file, calcium data (not electrical probes)

Data_processors: <-

- **Pull_and_process_data.py:** Extract target session info from AllenSDK API, process the spike information into a 2D Pandas dataframe that counts every unit's spikes for every time bin within a frame exposure window, for every frame exposure window. Exports the raw spike counts as a .pkl file to "output" folder.
    - **print_memory_usage:** diagnostics helper that prints the current Python process's RAM use.
    - **print_gpu_memory:** diagnostics helper that prints the current Python process's GPU RAM use and cached space in reserve.
    - **NeuronDataset:** turns "per-neuron spike time series + shared timing info" into on-demand binned spike vectors, so you can iterate or use a DataLoader(num_workers=..., batch_size=...) to parallelize/stream the per-neuron binning step without holding all processed neurons in memory at once.
        - **Process_neuron_wrapper:** Wrapper function to unpack arguments and call process_neuron.
        - **Process_neuron (modified)**: Outputs 1-D tensor vector that corresponds to a unit's spike count for each time ordered bin in which an image was shown.
    - **Create_directory_and_manifest:** Create the output directory and check or create the manifest.json file. Parameters: {directory_name (str): Name of the directory to be created. Default is 'output'.} Returns: {tuple: A tuple containing the path to the output directory and the path to the manifest.json file.}
    - **Create_cache_get_session_table:** Create an instance of the EcephysProjectCache class and get the session table. Parameters: {manifest_path (str): Path to the manifest.json file.} Returns: {tuple: A tuple containing the cache object and the session table.}
    - **Pick_session_and_pull_data (Modified):** Returns "spike_times" dictionary with unit label as key and the times it spikes as the value (list), and "stimulus_table", where each image presentation is a row and 2 columns are "start_time" and "stop_time" for when that image is shown. Prints additional attributes/methods of the session object.
        - Also returns: session filtered by the following parameters: (isi_violations_maximum=np.inf, amplitude_cutoff_maximum=np.inf,

presence_ratio_minimum=-np.inf,) (the same session used to get spike_times and stimulus_table)

- **Filter_valid_spike_times:** Filter the valid spike times using invalid times from the session object, converting the "spike_times" dic to "valid_spike_times"
- **Get_stimulus_table:** Retrieve the stimulus table for a given stimulus type.
- Calculate_bins: thin wrapper returning 4 items
- **Process_all_neurons (modified):** Process unit spike times into model ready binned spike count vectors for all neurons, returns CPU-stored list of 1-D tensor for each unit. Processes in parallel using batches of 1000.
    - If you intend to keep results on GPU for downstream ops, don't wrap with torch.tensor(...) at return; just return spike_matrix. If you want an actual 2-D tensor later, call torch.stack(spike_matrix) when needed (mind memory).
- **Create_and_prepare_spike_dataframe (modified):** Create a spike DataFrame and prepare it by adding a frame column.

  Parameters:
    - spike_matrix (torch.Tensor): The spike matrix containing processed spike times for all neurons. Is not directly the output of Process_all_neuronsbc that is a list.
        - Modified: spike_matrix (list): A list of 1000-neuron spike matrix batches containing processed spike times for all neurons.
    - spike_times (dict): A dictionary containing the original spike times.
    - stimulus_table (pd.DataFrame): The stimulus table containing stimulus-related data.
    - timesteps_per_frame (int): Number of timesteps per frame for binning.

  Returns:
    - pd.DataFrame: A prepared DataFrame containing spike data and a frame column.
- **Save_and_count_spike_dataframe:** Save the spike DataFrame to a pickle file in the specified directory and count the number of NaN values.

  Parameters:
    - spike_df (pd.DataFrame): The spike DataFrame to be saved.
    - session_number (int): The session number.
    - output_dir (str): The directory where the DataFrame should be saved.
    - timesteps_per_frame (int): Number of timesteps per frame for binning.

  Returns:
    - str: The name of the saved file.
- **Normalize_firing_rates:** Normalize the firing rates by calculating z-scores.

- **Filter_and_save_neurons:** ilter neurons based on firing rate thresholds, check for NaN values, and save the filtered data.
  Parameters:
    - normalized_firing_rates (pd.DataFrame): DataFrame containing normalized firing rates.
    - highest_value (float): Upper z-score threshold for filtering neurons.
    - lowest_value (float): Lower z-score threshold for filtering neurons.
    - session_number (int, optional): The session number, required if saving the data.
    - output_dir (str, optional): The directory where the DataFrame should be saved, required if saving the data.

  Returns:
    - pd.DataFrame: DataFrame containing filtered neurons.
- **Process_and_save_data:** Outdated
- **get_session_ids:** Get all available session IDs from the cache.
- **Process_session_batch:** Outdated
- **Process_all_sessions_in_batches:** Outdated
- **Master_function (modified):** Master function to execute the entire workflow from data extraction to saving. **Normalized firing rates functionality commented out.**
  Parameters:
    - session_number (int): The session number.
    - output_dir (str): The directory where data should be saved. Default is 'output'.
    - timesteps_per_frame (int, optional): Number of timesteps per frame for binning. Default is 10.

  Returns:
    - str: The name of the saved filtered data file.

- **Load_processed_data.py:** takes raw spike time .pkl file and normalizes and filters out units with abnormal spike counts (above 100 or below 0), as well as frames with values -1 (probably unnecessary). Exports cleaned .pkl file to “output” folder.
  - **check_file_exists (Modified):** Check if the pickle file exists in the specified directory.
    Parameters:
      - session_id (int): The ID of the session.
      - output_dir (str): The directory where the pickle file is expected to be found.
      - file_prefix (str): The prefix of the pickle file.

Returns:
    - bool: True if the file exists, False otherwise.
- **load_pickle_file:** Load a pickle file from the given path.
  Parameters:
    - pickle_path (str): Path to the pickle file.
  Returns:
    - object: The object loaded from the pickle file, or None if loading fails.
- **save_pickle_file:** Save data to a pickle file at the given path.
  Parameters:
    - data (object): The data to save.
    - pickle_path (str): Path to save the pickle file.
- **clean_avg_firing_rates:** Clean the average firing rates in the given DataFrame.
  Parameters:
    - df (DataFrame): The DataFrame with the firing rates data.
    - highest_value (float): Upper z-score threshold for filtering neurons.
    - lowest_value (float): Lower z-score threshold for filtering neurons.
  Returns:
    - DataFrame: The cleaned DataFrame.
- **master_cleaning_and_saving(Modified):** Master function to check, clean, and save the firing rate data within an 'output' directory. Remove line removing rows with -1 frame values? (no reason why this would happen)?
  Parameters:
    - session_id (int): The ID of the session to be processed.
    - output_dir (str): The directory where the pickle file is located.
    - original_pickle_prefix (str): The prefix of the original pickle file.
    - new_pickle_prefix (str): The prefix for the new cleaned pickle file.
    - highest_value (float): The threshold to remove neurons with a high z-score.
    - lowest_value (float): The threshold to remove neurons with a z-score never exceeding this value.

- **data_splitter.py:** It prepares a time-ordered dataframe (last column = class) for sequence models by label-encoding the targets, dropping zero-variance features, splitting (no shuffle) into train/val/test, segmenting into fixed-length sequences, and reshaping to tensors of shape (B, T, N, 1) while using the label from the last timestep of each sequence. It then builds PyTorch DataLoaders for each split and includes a helper to compute a feature correlation matrix.
    - **DataSplitter (class):**

- **__innit__:** initialize X and y attributes using a further processed session pandas dataframe (remove_0_columns, _prepare_data)
- **_prepare_data:** split session data pandas dataframe into X and y, convert semantic y labels to integers.
- **Remove_0_columns:** remove units that contain zero information, i.e don't vary in their spikes at all (inactive)
- **_split_and_reshape:** reshape feature pandas dataframe into 3-D (with an added 4th dimension for CNN compatibility). First dimension is each individual sample, i.e each time an image was shown. Each of these is associated with a 2-D matrix of the 2nd and 3rd dimension, which is each time ordered bin and each unit. Label pandas dataframe is reshaped from one label per time bin to one label per sample (with an added 2nd dimension for CNN compatibility)
- Potential issue: "[:num_samples * self.seq_len]" safety check is flawed because it only solves issues at the tail. If corruption happened in the beginning or middle, this filter will corrupt data.
- **_create_loaders (Modified):** "Master function" of class, all previous helper functions utilized to create cleaned, batched, data loader class attributes for train, test, split (as well as the raw X/y tensor splits. X and y are still pandas dataframes but X_train etc. are pytorch tensors).
    - Shuffle batches set to True for train loader to reduce potential overfitting.
- **Compute_correlation_matrix:** computes the correlation matrix of the units with each other.

Models:

Old Files: irrelevant

Output: Extracted and Processed data/artifacts used by models

- **Session_#:** folder containing that session's .nwb file
- **Channels.csv:** contains individual channel ids from the probes and their spatial location
- **Manifest.json:** configurations for fetching data from SDK
- **Probes.csv:** probe id's and technical specifications
- **Session_details.csv:** LSTM-specific training run hyperparameters and results
    - Session id's do not line up with .nwb or .pkl files in output parent folder

    - Some train_acc values are > 1 (e.g., 1.12, 1.37) while others look like percentages (92.69) — this suggests a units/logging inconsistency (fraction vs percent). You may want to standardize the logging.
- **Sessions.csv:** largely irrelevant metadata about the session and it's mouse
- **Spike_trains_plot.png:** visualization of spike data
- **Spike_trains_with_stimulus_session_(session_id)_(bins).pkl:**
    - Session pkl's do not line up 1-1 with session .nwb files.
    - your code has hard-coded/changed output_dir paths at times, so artifacts may have landed in different folders on different runs.
    - Only the 250 ms stimulus frames are included, binned into a set number of intervals, with one vector per interval.
    - Number after session id corresponds to the timesteps_per_frame parameter (number of intervals) in the processing code (pull_and_process_data.py / process_and_save_data).
    - Each vector has a "frame" feature (last), and many unit features (activations for each neuron, as ints).
    - Each unit feature's data is a int count of many times the neuron spiked within the time frame of the current bin (row)
    - Rows (bins) time ordered. One training example corresponds to a specific frame's block of time bins during which that frame was shown. A single frame will have multiple training example blocks because they were shown multiple times (roughly 50)
- **Units.csv:** metadata about unit quality and physical characteristics

Pictures: high level visualizations of a couple model information flows

Results:

- **3_shot_(model type)_per_mouse.csv:** For each image (frame .pkl feature), 3 of its time blocks are randomly selected and used to train the model, instead of the full 50. Model hyperparameter and performance reported.
    - Should be per session, not mouse, one mouse can have multiple sessions
- **highest_test_accuracy.png:** Best LSTM accuracy (best model) per mouse session
    - Title slightly misleading one mouse can have multiple sessions
- **Highest_test_accuracy_no_subs.png:** ^

Saved_models:

Wandb: "weights and biases" folder to track performance, hyperparameters, env of history of training run. Hyperparameters and weights and biases saved elsewhere so safe to delete, folder is just for training run evaluation

- **Run2024…7j:** Static ST-GAT training run
    - **Files:**
        - **Config.yaml:** hyperparamters and env info of training run
        - **Output.log:** per epoch data
        - **Requirements.txt:** Python environment snapshot: a pinned list of packages (with exact versions) needed to reproduce run
        - **Wandb-summary.json:** Most recent snapshot of performance called at a specific epoch.
    - **Logs**: Debugging information