1. Environment Setup and IBL Data Access

This section initializes the necessary Python libraries and establishes the connection to the International Brain Laboratory (IBL) public data repository using the `ONE` API. We also initialize the `AllenAtlas` for anatomical referencing.

Technical Dependencies

The following code block imports all core libraries required for data loading, electrophysiology processing, statistical analysis, and visualization.

In [ ]:
```
!pip install ONE-api
!pip install ibllib
!apt-get -qq install -y libfluidsynth1
from one.api import ONE
from brainbox.io.one import SpikeSortingLoader
from brainbox.singlecell import calculate_peths
from iblatlas.atlas import AllenAtlas

# Computing and plotting libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import shapiro, normaltest, mannwhitneyu, zscore, sem
import statsmodels.formula.api as smf
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.multitest import multipletests

# Create ONE and Atlas instances
one = ONE(base_url='https://openalyx.internationalbrainlab.org', password
ba = AllenAtlas()
```

2. Filtering by Region of Interest (VTA)

This step queries the IBL database to identify all recording sessions containing electrode insertions within the Ventral Tegmental Area (VTA), based on the Allen Atlas coordinates.

The loop then iterates through these sessions to load the spike sorting and behavioral data, calculating a preliminary summary of available VTA neurons and correct trials for quality control.

In [ ]:
```
roi = "VTA" # Region Of Interest (acronym according to Allen Atlas)
ses = one.alyx.rest('insertions', 'list', atlas_acronym = roi)
print(f'{len(ses)} recordings from the VTA')
eids = [i['session'] for i in ses]
pids = [i['id'] for i in ses]
```

```
summary = []

for pid, eid in zip(pids, eids):
    try:
        # load spikes, clusters, and trial info
        sl = SpikeSortingLoader(pid=pid, one=one, atlas=ba)
        spikes, clusters, channels = sl.load_spike_sorting()
        clusters = sl.merge_clusters(spikes, clusters, channels)
        trials = one.load_object(eid, 'trials')

        # compute neuron counts
        total_clusters = len(clusters['cluster_id'])
        good_clusters = np.sum(clusters['label'] == 1)

        vta_mask = np.array([roi in acr for acr in clusters['acronym']])
        good_mask = clusters['label'] == 1
        good_vta_neurons = np.sum(vta_mask & good_mask)
        total_vta_neurons = np.sum(vta_mask)

        # compute correct trial count
        correct_trials = np.sum(trials['feedbackType'] == 1)

        # summary
        summary.append({
            'pid': pid,
            'eid': eid,
            'total_vta_neurons': total_vta_neurons,
            'good_vta_neurons': good_vta_neurons,
            'total_correct_trials': int(correct_trials)
        })

    except Exception as e:
        print(f"Failed to load session {pid}: {e}")

# Convert to DataFrame
import pandas as pd
df = pd.DataFrame(summary)
print(df)
```

3. Final Session Selection and Data Loading

Due to the limited number of high-quality VTA recordings (N=8 total), a single session was selected based on three criteria to ensure the reliability of the single-unit analysis: High count of VTA neurons, high firing stability, and sufficient number of correct trials.

Session '50ebb677-e4a3-4421-b74f-1997a9cd1ad1' was chosen for detailed analysis.

In [ ]:
```
pid = '50ebb677-e4a3-4421-b74f-1997a9cd1ad1'
eid, _ = one.pid2eid(pid)

# Load sorting data
sl = SpikeSortingLoader(pid=pid, one=one, atlas=ba)
```

```
spikes, clusters, channels = sl.load_spike_sorting()
clusters = sl.merge_clusters(spikes, clusters, channels)

# Load trial data
trials = one.load_object(eid, 'trials')
print(f"Total trials: {len(trials['goCue_times'])}")

for i, acronym in enumerate(clusters['acronym']):
    print(f"{i}: {acronym}")
```

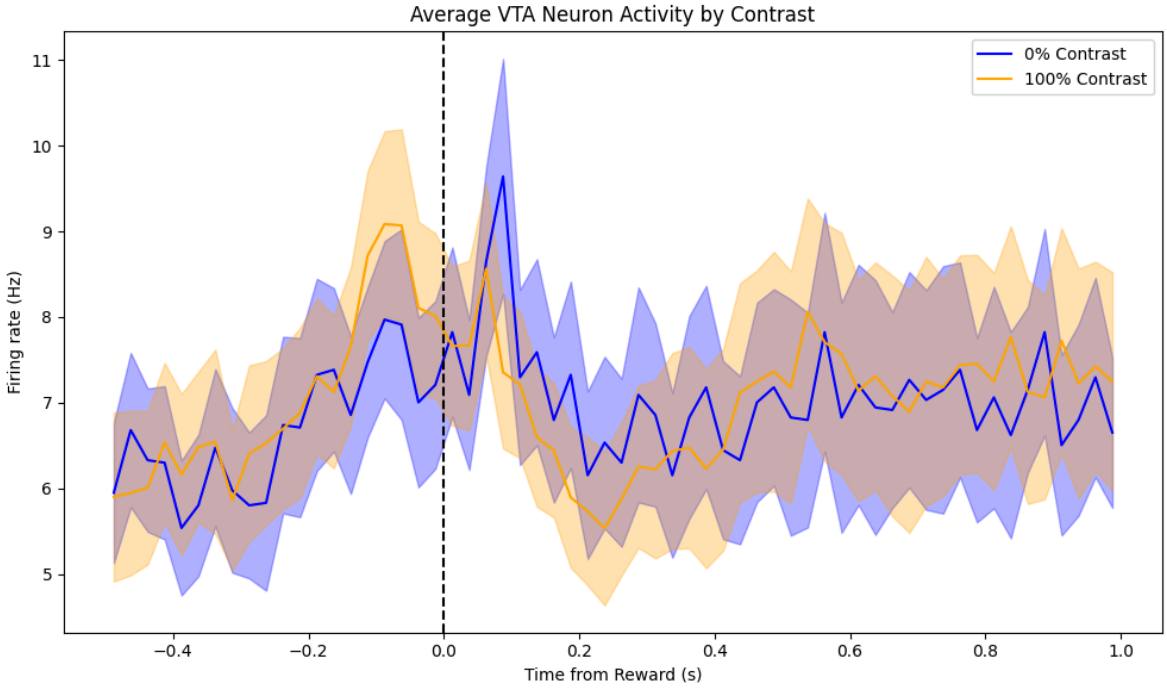
4. Peristimulus Time Histogram (PETH) Calculation

This section defines the alignment times (reward feedback) for the two main conditions (High Contrast/Low Uncertainty vs. Low Contrast/High Uncertainty) and computes the mean PETHs (firing rate) across all VTA neurons for a preliminary comparison of the reward response profiles.

In [4]:

```
## GLOBAL TIME PARAMETERS
# Time parameters
T_BEFORE = 0.5
T_AFTER = 1.0
BIN_SIZE = 0.025

# Time parameteres for PETHS & graphs
T_BINS = np.arange(-T_BEFORE, T_AFTER + BIN_SIZE, BIN_SIZE)
TIME_CENTERS = T_BINS[:-1] + BIN_SIZE / 2

# Identify all VTA neurons
vta_neurons = np.where(['VTA' in ac for ac in clusters['acronym']])[0]
vta_neuron_ids = np.unique(spikes['clusters'][np.isin(spikes['clusters'],
rewarded = trials['feedbackType'] == 1

# Get contrast (combine left and right where NaN)
contrast = trials['contrastLeft'].copy()
nan_mask = np.isnan(contrast)
contrast[nan_mask] = trials['contrastRight'][nan_mask]

# 0% and 100% contrast trials
low_contrast_trials = rewarded & (contrast == 0.0)
high_contrast_trials = rewarded & (contrast == 1.0)

# Alignment times for low and high contrast rewarded trials
align_times_low = trials['feedback_times'][low_contrast_trials]
align_times_high = trials['feedback_times'][high_contrast_trials]

def compute_peth(spike_times, spike_clusters, neuron_ids, align_times,T_B
    """Compute PETH firing rates for each neuron aligned to events."""
    peth_all = np.zeros((len(neuron_ids), len(T_BINS)-1))
    for i, neuron in enumerate(neuron_ids):
        # Spike times for this neuron
        neuron_spikes = spike_times[spike_clusters == neuron]

        # Collect spike counts for all events
        counts_all = []
        for t in align_times:
            # Relative spike times to event
            rel_spikes = neuron_spikes - t
```

```
            # Histogram counts in bins
            counts, _ = np.histogram(rel_spikes, T_BINS)
            counts_all.append(counts)

        # Average counts per bin across trials, convert to firing rate (H
        mean_counts = np.mean(counts_all, axis=0)
        firing_rate = mean_counts / BIN_SIZE  # spikes per second
        peth_all[i, :] = firing_rate
    return peth_all

# Compute PETHs
peth_low = compute_peth(spikes['times'], spikes['clusters'], vta_neuron_i
peth_high = compute_peth(spikes['times'], spikes['clusters'], vta_neuron_

# Average and SEM across neurons
mean_low = np.mean(peth_low, axis=0)
sem_low = np.std(peth_low, axis=0) / np.sqrt(len(vta_neuron_ids))

mean_high = np.mean(peth_high, axis=0)
sem_high = np.std(peth_high, axis=0) / np.sqrt(len(vta_neuron_ids))

# Time axis for plotting (bin centers)
tscale = T_BINS[:-1] + BIN_SIZE / 2

# Plot
plt.figure(figsize=(10,6))
plt.fill_between(tscale, mean_low - sem_low, mean_low + sem_low, color='b
plt.plot(tscale, mean_low, color='blue', label='0% Contrast')
plt.fill_between(tscale, mean_high - sem_high, mean_high + sem_high, colo
plt.plot(tscale, mean_high, color='orange', label='100% Contrast')
plt.axvline(0, linestyle='--', color='black')
plt.xlabel('Time from Reward (s)')
plt.ylabel('Firing rate (Hz)')
plt.title('Average VTA Neuron Activity by Contrast')
plt.legend()
plt.tight_layout()
plt.show()
```



In this average plot, there is a trend where high-contrast trials show increased activity prior to the reward, possibly reflecting an anticipatory signal for an expected reward. A second, smaller peak is observed following the reward. In contrast, in the low-contrast correct trials, pre-reward activity is lower, while a more prominent peak emerges after the reward, potentially indicating a response to unexpected reward delivery. However, without statistical analysis, these interpretations remain speculative.

In [ ]:
```
trial_window = T_BEFORE + T_AFTER

#Collect trial-by-trial firing rates
trial_rates_low = []
trial_rates_high = []

for neuron_id in vta_neuron_ids:
    neuron_spikes = spikes['times'][spikes['clusters'] == neuron_id]

    for t in align_times_low:
        rel_spikes = neuron_spikes - t
        spikes_in_window = rel_spikes[(rel_spikes >= -T_BEFORE) & (rel_sp
        firing_rate = len(spikes_in_window) / trial_window
        trial_rates_low.append(firing_rate)

    for t in align_times_high:
        rel_spikes = neuron_spikes - t
        spikes_in_window = rel_spikes[(rel_spikes >= -T_BEFORE) & (rel_sp
        firing_rate = len(spikes_in_window) / trial_window
        trial_rates_high.append(firing_rate)

#Descriptive statistics
def describe_trials(data, label):
    print(f"\n Descriptive stats for {label}")
    print(f"Mean: {np.mean(data):.3f}")
    print(f"Median: {np.median(data):.3f}")
    print(f"Std: {np.std(data):.3f}")
    print(f"N (trials): {len(data)}")
    print(f"Shapiro-Wilk p-value: {shapiro(data).pvalue:.4f}")
    print(f"D'Agostino K² p-value: {normaltest(data).pvalue:.4f}")
describe_trials(trial_rates_low, "Low Contrast (Correct)")
describe_trials(trial_rates_high, "High Contrast (Correct)")
#plot
plt.figure(figsize=(10, 5))
sns.histplot(trial_rates_low, kde=True, color='blue', label='Low Contrast
sns.histplot(trial_rates_high, kde=True, color='green', label='High Contr
plt.axvline(np.mean(trial_rates_low), linestyle='--', color='blue', label
plt.axvline(np.mean(trial_rates_high), linestyle='--', color='green', lab
plt.title("Trial-by-Trial VTA Firing Rates\n(Reward Aligned, Correct Tria
plt.xlabel("Firing Rate (Hz)")
plt.ylabel("Density")o
plt.legend()
plt.tight_layout()
plt.show()
```

In this session, there are 39 VTA neurons and 583 total trials. Among the correct trials, 35 are low contrast and 134 are high contrast. For the low contrast condition,

the mean firing rate is 6.94 Hz (SD = 8.31), while for the high contrast condition it's slightly higher at 7.04 Hz (SD = 8.79). Both Shapiro-Wilk and D'Agostino tests indicate that the firing rate distributions significantly deviate from normality ($p < 0.0001$) in both contrast conditions.

Reward Modulation Analysis:

I aim to identify VTA neurons that are responsive to reward delivery. Specifically, I test whether each neuron's firing rate significantly increases or decreases following the reward compared to a pre-reward baseline.

To do this, I compute the firing rate of each neuron during two time windows:

- A baseline window before reward delivery (−1.0 to −0.5 seconds). The interval from −0.5 to 0 seconds is excluded due to the possibility of anticipatory responses before the reward.
- A post-reward window immediately after reward delivery (0 to +0.5 seconds). The Mann-Whitney U test is then used to compare the distributions of firing rates between these two periods across all correct trials for each neuron. This allows the detection of neurons whose activity is significantly modulated by the presence of a reward, regardless of stimulus contrast.

Why do we do this?

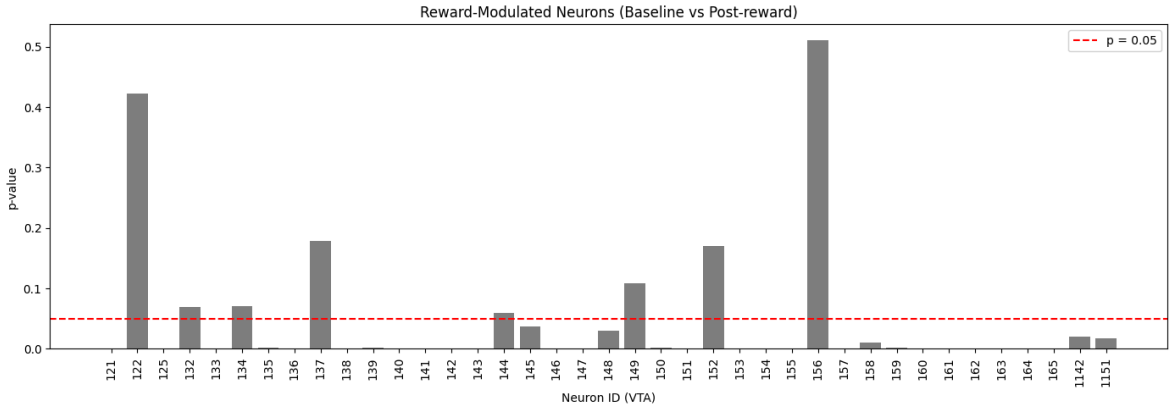
The original hypothesis focuses on how dopaminergic neurons in the VTA encode reward prediction errors under varying levels of sensory uncertainty. However, analyzing the entire VTA population may obscure the signal, as not all VTA neurons are dopaminergic or reward-sensitive. Results are visualized as a p-value bar graph and a z-scored PSTH heatmap, with significantly modulated neurons at the top.

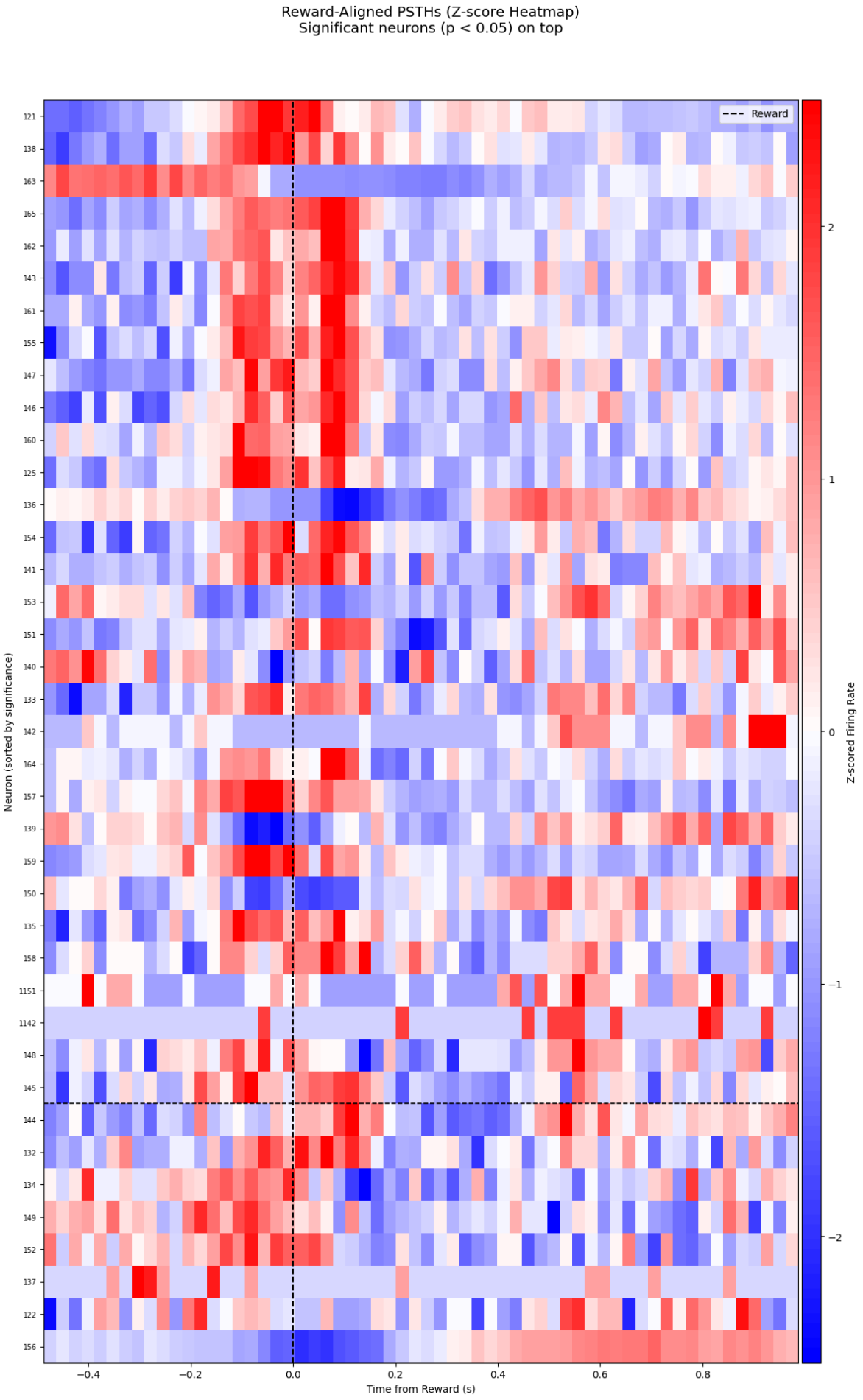
In the bar plot we can see that the non-responsive neurons are firing very different than the majority of neurons. Majority of the neurons dont have a visible bar in the plot. Note: The neurons 145, 148, 158, 1142, 1151 and 135, 150 and 139 (slighlty) are passing but still noted because there is a visual detectable difference in the firing rates in the bar graph. When we check the heatmap these neurons also at the bottom of the heatmap's significance line.

In [7]:
```
# parameters
baseline_window = (-1.0, -0.5)
post_window = (0.0, 0.5)
analysis_window = (-1.0, 1.0)

# Results
p_values = []
neuron_ids = []
psth_matrix = []

for neuron_id in vta_neuron_ids:
    neuron_spikes = spikes['times'][spikes['clusters'] == neuron_id]
```

```
    trial_histograms = []

    baseline_rates = []
    post_rates = []

    for rt in reward_times:
        aligned_spikes = neuron_spikes - rt
        hist, _ = np.histogram(aligned_spikes, bins=T_BINS)
        trial_histograms.append(hist)

        baseline_spikes = np.sum((aligned_spikes >= baseline_window[0]) &
        post_spikes = np.sum((aligned_spikes >= post_window[0]) & (aligne
        baseline_rates.append(baseline_spikes / (baseline_window[1] - bas
        post_rates.append(post_spikes / (post_window[1] - post_window[0])

    if len(baseline_rates) > 5 and len(post_rates) > 5:
        stat, p = mannwhitneyu(baseline_rates, post_rates, alternative='t
        p_values.append(p)
        neuron_ids.append(neuron_id)
        psth_matrix.append(np.mean(trial_histograms, axis=0) / BIN_SIZE)

# Plot p-value bar graph
plt.figure(figsize=(14, 5))
bars = plt.bar(range(len(p_values)), p_values, color='gray')
plt.axhline(y=0.05, color='red', linestyle='--', label='p = 0.05')
plt.xticks(range(len(p_values)), neuron_ids, rotation=90)
plt.ylabel('p-value')
plt.xlabel('Neuron ID (VTA)')
plt.title('Reward-Modulated Neurons (Baseline vs Post-reward)')
plt.legend()
plt.tight_layout()
plt.show()

# Z-score PSTHs and plot heatmap
p_values = np.array(p_values)
neuron_ids = np.array(neuron_ids)
psth_matrix = np.array(psth_matrix)

baseline_mask = (TIME_CENTERS >= baseline_window[0]) & (TIME_CENTERS < ba
z_psth_matrix = zscore(psth_matrix, axis=1)

sorted_idx = np.argsort(p_values)
sorted_z = z_psth_matrix[sorted_idx]
sorted_ids = neuron_ids[sorted_idx]
cutoff = np.sum(p_values < 0.05)

fig, ax = plt.subplots(figsize=(12, 0.4 * len(neuron_ids) + 4))
im = ax.imshow(sorted_z, aspect='auto', cmap='bwr',
               extent=[TIME_CENTERS[0], TIME_CENTERS[-1], len(neuron_ids)
               vmin=-2.5, vmax=2.5)

# Smaller colorbar
from mpl_toolkits.axes_grid1 import make_axes_locatable
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="2.5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Z-scored Firing Rate')

if cutoff > 0:
    ax.axhline(cutoff, color='black', linestyle='--', linewidth=1.2)
```

```
ax.axvline(0, linestyle='--', color='k', label='Reward')
ax.set_xlabel('Time from Reward (s)')
ax.set_ylabel('Neuron (sorted by significance)')
fig.suptitle('Reward-Aligned PSTHs (Z-score Heatmap)\nSignificant neurons
ax.set_yticks(np.arange(len(neuron_ids)) + 0.5)
ax.set_yticklabels(sorted_ids, fontsize=7)
ax.legend()
plt.tight_layout()
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

# significant neuron id
significant_neurons = neuron_ids[p_values < 0.05]
print("Reward-modulated neurons:", significant_neurons.tolist())
```






```
Reward-modulated neurons: [121, 125, 133, 135, 136, 138, 139, 140, 141, 14
2, 143, 145, 146, 147, 148, 150, 151, 153, 154, 155, 157, 158, 159, 160, 1
61, 162, 163, 164, 165, 1142, 1151]
```

There is a risk of %5 false positive. What is my error rate? FDR?

In [8]:
```
# FDR correction
rej, pvals_fdr, _, _ = multipletests(p_values, alpha=0.05, method='fdr_bh

# Number of significant neurons BEFORE and AFTER FDR
n_raw_sig = np.sum(p_values < 0.05)
n_fdr_sig = np.sum(rej)

print(f"Raw significant neurons (p < 0.05): {n_raw_sig}")
print(f"FDR-corrected significant neurons: {n_fdr_sig}")
print(f"Estimated false positive rate: {(n_raw_sig - n_fdr_sig) / max(n_r
```

```
Raw significant neurons (p < 0.05): 31
FDR-corrected significant neurons: 31
Estimated false positive rate: 0.00
```

All 31 neurons survived the FDR correction!

5. Trial-by-Trial Analysis and Mixed-Effects Modeling

To quantify the difference in uncertainty-modulated reward responses, the average firing rate for each trial and condition is calculated. Statistical distributions are visualized, and a Mixed-Effects Model (LME) is used to formally test the effect of 'contrast' on the 'firing_rate' while controlling for inter-neuron variability.

In [9]:
```
trial_window = T_BEFORE + T_AFTER  # Duration in seconds

# Prepare lists for DataFrame rows
data = {
    'firing_rate': [],
    'contrast': [],
    'neuron': []
}

# Loop through the selected neurons only
for neuron_id in vta_neuron_ids:
    neuron_spikes = spikes['times'][spikes['clusters'] == neuron_id]

    # Low contrast trials
    for t in align_times_low:
        rel_spikes = neuron_spikes - t
        spikes_in_window = rel_spikes[(rel_spikes >= -T_BEFORE) & (rel_sp
        firing_rate = len(spikes_in_window) / trial_window

        data['firing_rate'].append(firing_rate)
        data['contrast'].append('low')
        data['neuron'].append(str(neuron_id))

    # High contrast trials
    for t in align_times_high:
        rel_spikes = neuron_spikes - t
        spikes_in_window = rel_spikes[(rel_spikes >= -T_BEFORE) & (rel_sp
```

```
        firing_rate = len(spikes_in_window) / trial_window

        data['firing_rate'].append(firing_rate)
        data['contrast'].append('high')
        data['neuron'].append(str(neuron_id))

# Create DataFrame
df = pd.DataFrame(data)

# Fit mixed-effects model: firing_rate ~ contrast + (1|neuron)
model = smf.mixedlm("firing_rate ~ contrast", df, groups=df["neuron"])
result = model.fit()
print(result.summary())
```

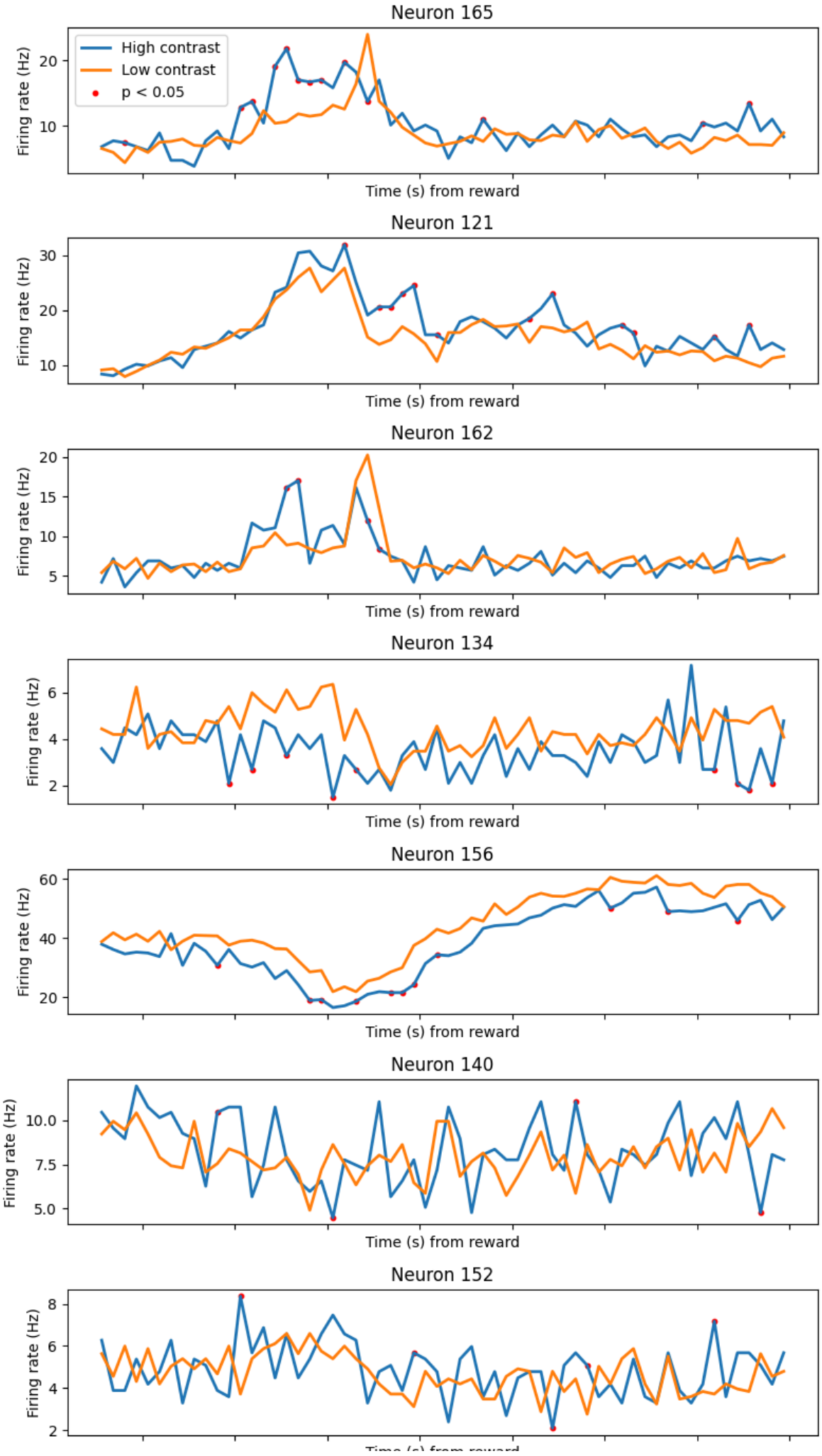
```
          Mixed Linear Model Regression Results
============================================================
Model:             MixedLM Dependent Variable: firing_rate
No. Observations:  6591    Method:             REML
No. Groups:        39      Scale:              36.3221
Min. group size:   169     Log-Likelihood:     -21292.0602
Max. group size:   169     Converged:          Yes
Mean group size:   169.0
------------------------------------------------------------
                Coef.  Std.Err.   z    P>|z| [0.025 0.975]
------------------------------------------------------------
Intercept        7.041    1.020  6.906 0.000  5.043  9.040
contrast[T.low] -0.103    0.183 -0.560 0.575 -0.462  0.256
Group Var       40.274    1.543
============================================================
```

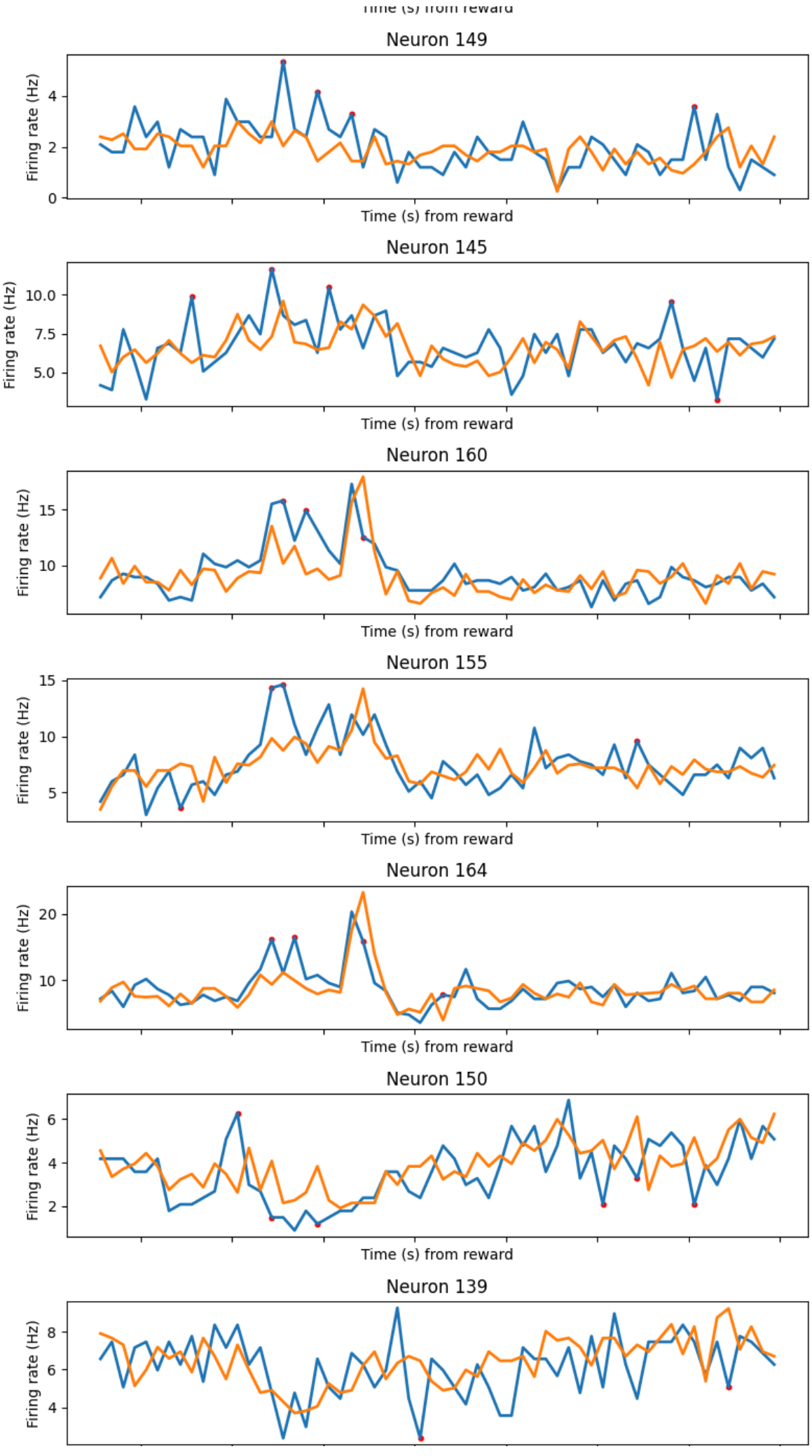
The LME shows that contrast level (high vs. low) does not significantly predict firing rate (p = 0.859), while overall baseline firing (intercept) is significant. This suggests that contrast alone doesn't explain variation in VTA neuron activity across trials, but the population signal may be lost by the inclusion of potential non-reward-responsive neurons.

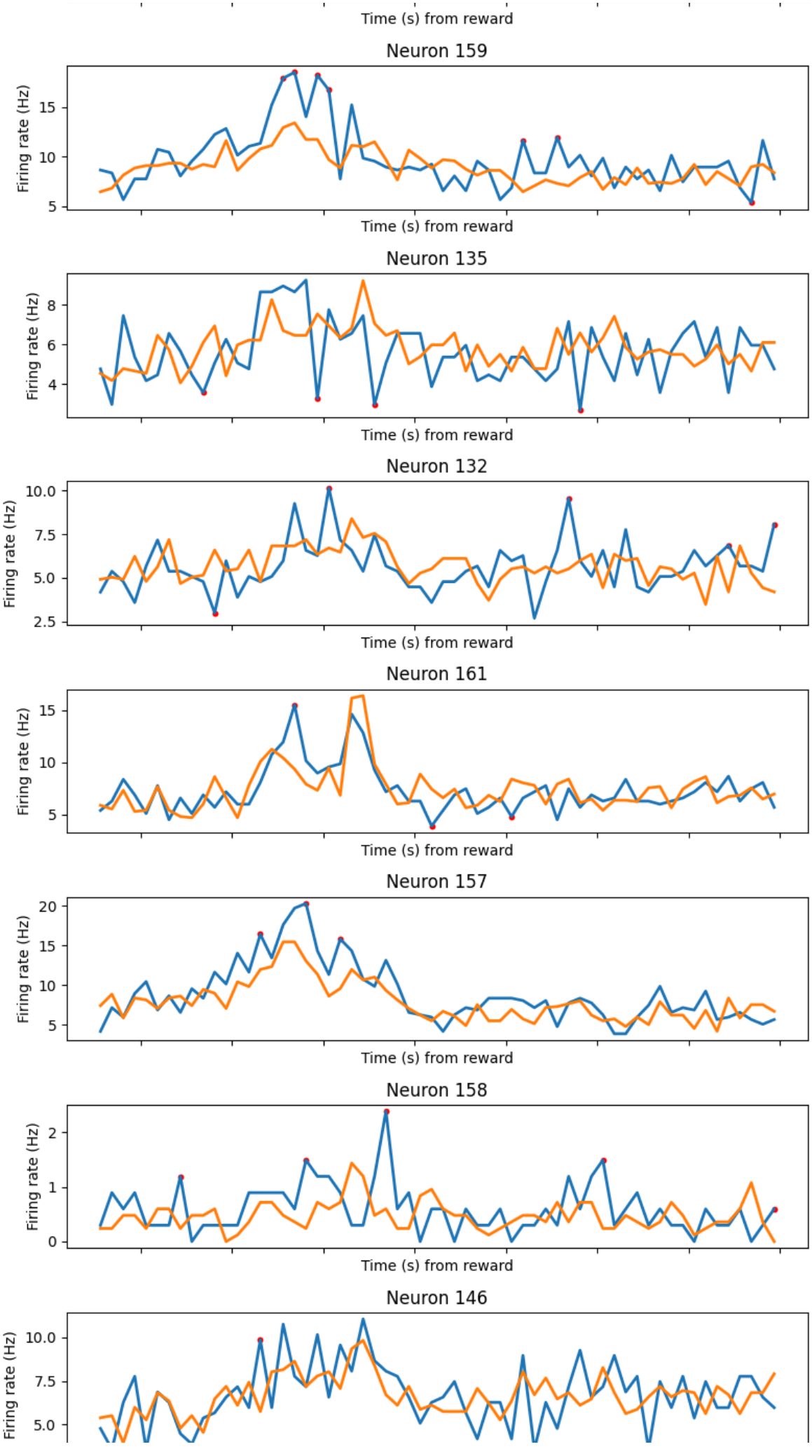
Instead of population level analysis we will focus on individual level analysis. For each time bin and each neuron, a Mann-Whitney U test is performed and the significant points are marked in average plot in each neuron.

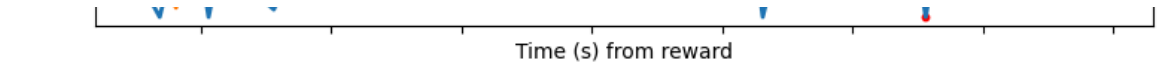
In [10]:
```
#Initialize storage
pvals_by_neuron = {}
firing_by_neuron = {}
min_pval_by_neuron = {}

# Loop through neurons
for neuron_id in vta_neuron_ids:
    neuron_spikes = spikes['times'][spikes['clusters'] == neuron_id]

    high_matrix, low_matrix = [], []

    for i, rt in enumerate(reward_times):
        rel_spikes = neuron_spikes - rt
        spike_counts, _ = np.histogram(rel_spikes, bins=T_BINS)
```

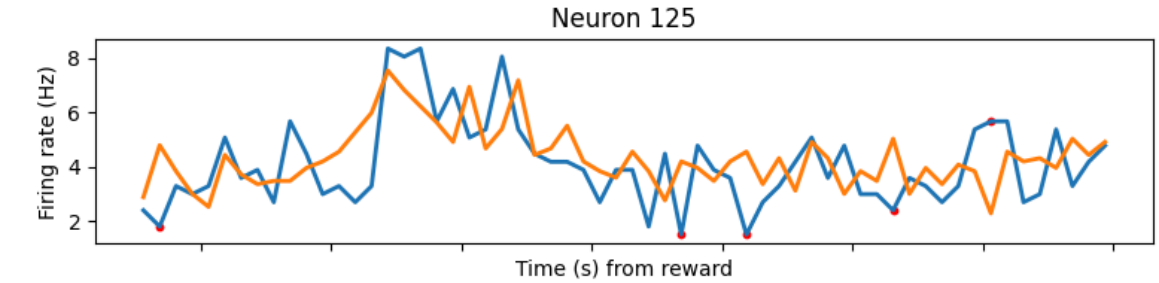
```
        if contrast_labels[i] == 'high':
            high_matrix.append(spike_counts)
        elif contrast_labels[i] == 'low':
            low_matrix.append(spike_counts)

    high_matrix = np.array(high_matrix)
    low_matrix = np.array(low_matrix)

    if high_matrix.shape[0] >= 5 and low_matrix.shape[0] >= 5:
        pvals = []
        for b in range(len(TIME_CENTERS)):
            try:
                _, p = mannwhitneyu(high_matrix[:, b], low_matrix[:, b],
            except ValueError:
                p = 1.0
            pvals.append(p)
        pvals = np.array(pvals)
        pvals_by_neuron[neuron_id] = pvals
        firing_by_neuron[neuron_id] = {
            'high': high_matrix.mean(axis=0) / BIN_SIZE,
            'low': low_matrix.mean(axis=0) / BIN_SIZE
        }
        min_pval_by_neuron[neuron_id] = np.min(pvals)

# Sort neurons by their most significant p-value
sorted_neuron_ids = sorted(min_pval_by_neuron, key=min_pval_by_neuron.get

# Plotting
num_neurons = len(sorted_neuron_ids)
fig, axes = plt.subplots(num_neurons, 1, figsize=(8, 2 * num_neurons), sh

if num_neurons == 1:
    axes = [axes]

for idx, neuron_id in enumerate(sorted_neuron_ids):
    pvals = pvals_by_neuron[neuron_id]
    fr_high = firing_by_neuron[neuron_id]['high']
    fr_low = firing_by_neuron[neuron_id]['low']

    ax = axes[idx]
    ax.plot(TIME_CENTERS, fr_high, label='High contrast', linewidth=2)
    ax.plot(TIME_CENTERS, fr_low, label='Low contrast', linewidth=2)

    sig_mask = np.array(pvals) < 0.05
    ax.scatter(TIME_CENTERS[sig_mask], fr_high[sig_mask], color='red', s=
    ax.set_title(f'Neuron {neuron_id}')
    ax.set_ylabel('Firing rate (Hz)')
    ax.set_xlabel('Time (s) from reward')

axes[0].legend()
plt.tight_layout()
plt.show()
```

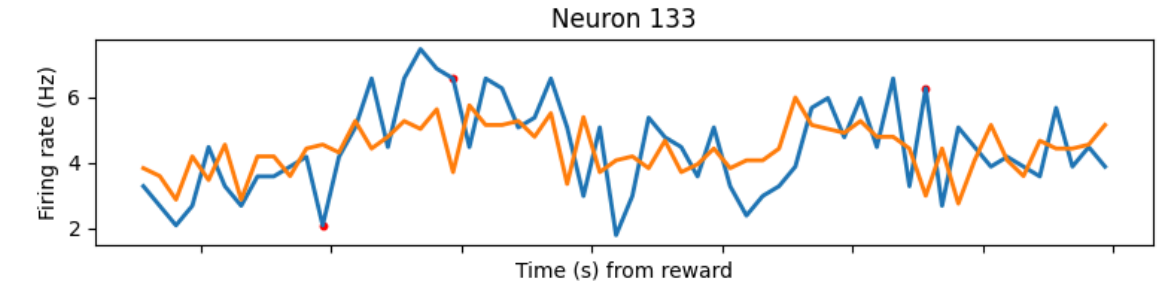

Neuron 165
High contrast
Low contrast
p < 0.05
Firing rate (Hz)
Time (s) from reward
Neuron 121
Neuron 162
Neuron 134
Neuron 156
Neuron 140
Neuron 152

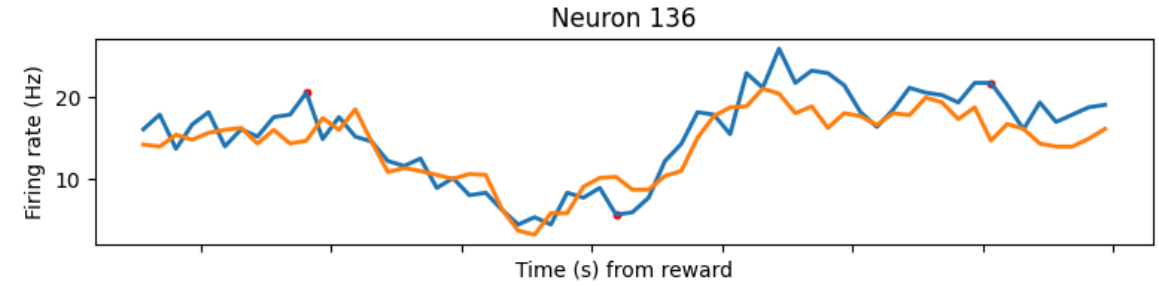

Time (s) from reward
Neuron 149
Firing rate (Hz)
Time (s) from reward
Neuron 145
Firing rate (Hz)
Time (s) from reward
Neuron 160
Firing rate (Hz)
Time (s) from reward
Neuron 155
Firing rate (Hz)
Time (s) from reward
Neuron 164
Firing rate (Hz)
Time (s) from reward
Neuron 150
Firing rate (Hz)
Time (s) from reward
Neuron 139
Firing rate (Hz)

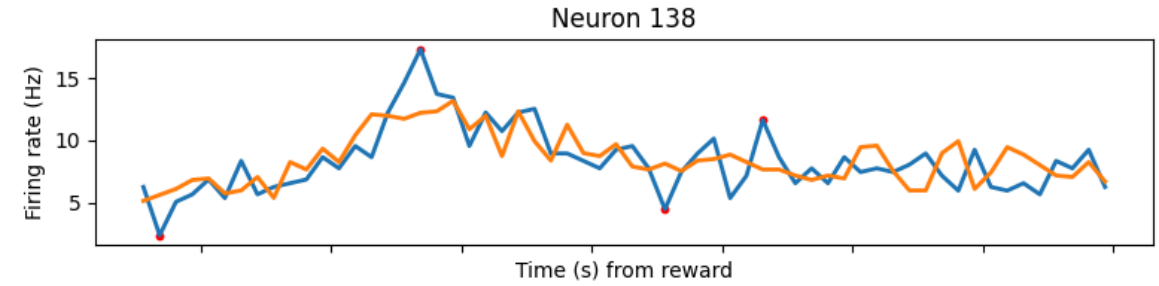

Time (s) from reward
Neuron 159
Firing rate (Hz)
15
10
5
Time (s) from reward
Neuron 135
Firing rate (Hz)
8
6
4
Time (s) from reward
Neuron 132
Firing rate (Hz)
10.0
7.5
5.0
2.5
Time (s) from reward
Neuron 161
Firing rate (Hz)
15
10
5
Time (s) from reward
Neuron 157
Firing rate (Hz)
20
15
10
5
Time (s) from reward
Neuron 158
Firing rate (Hz)
2
1
0
Time (s) from reward
Neuron 146
Firing rate (Hz)
10.0
7.5
5.0

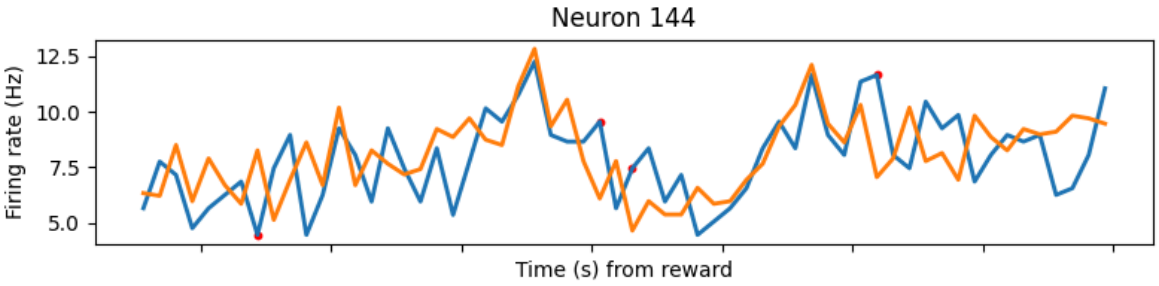

Time (s) from reward

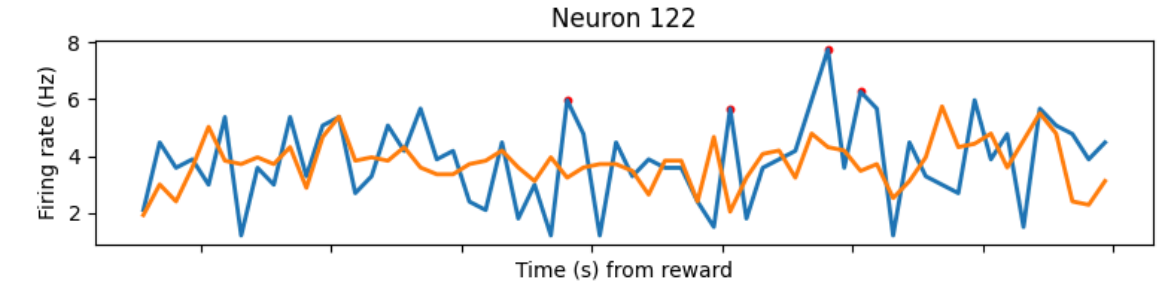

Neuron 125
Firing rate (Hz)
8
6
4
2
Time (s) from reward

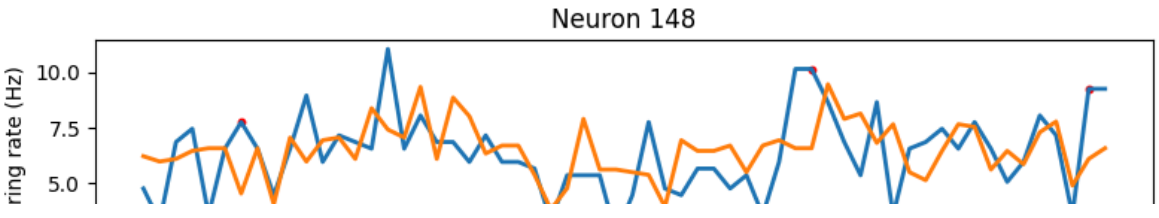

Neuron 133
Firing rate (Hz)
6
4
2
Time (s) from reward

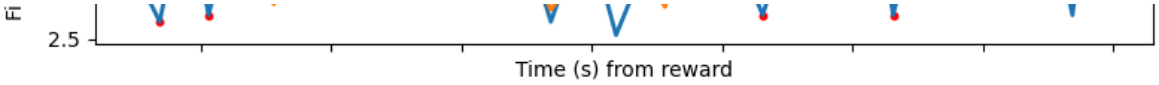

Neuron 136
Firing rate (Hz)
20
10
Time (s) from reward

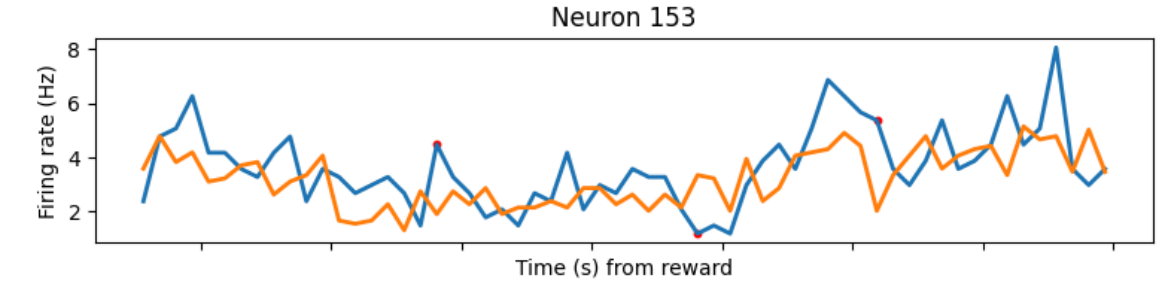

Neuron 138
Firing rate (Hz)
15
10
5
Time (s) from reward

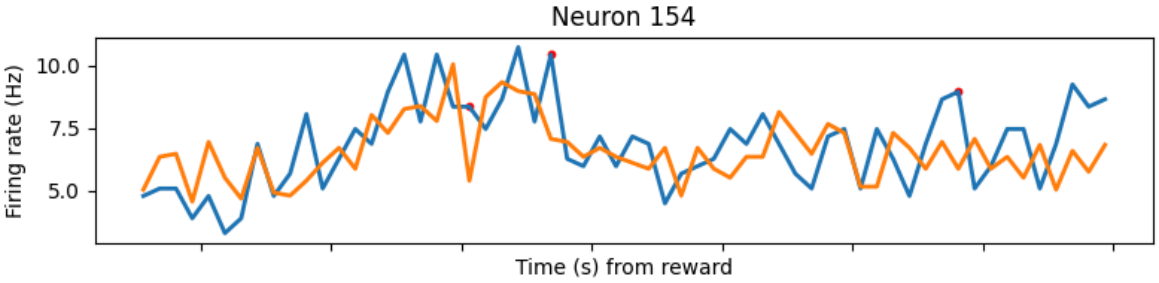

Neuron 144
Firing rate (Hz)
12.5
10.0
7.5
5.0
Time (s) from reward

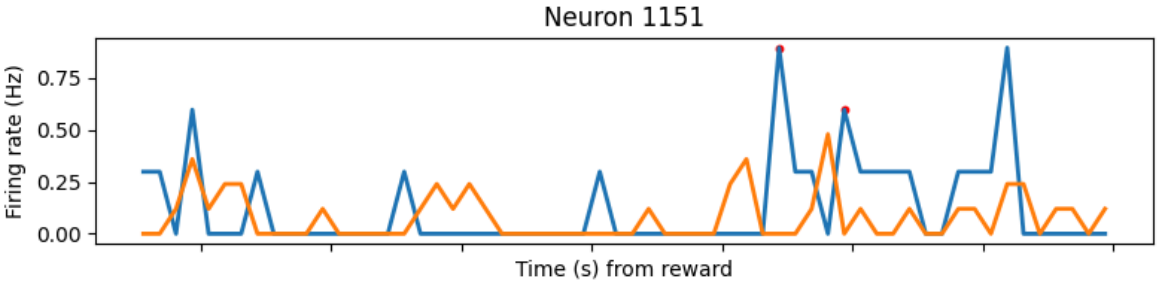

Neuron 122
Firing rate (Hz)
8
6
4
2
Time (s) from reward

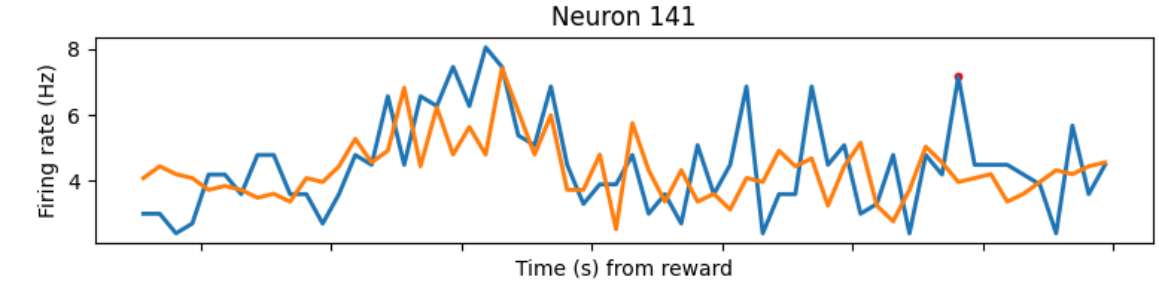

Neuron 148
ring rate (Hz)
10.0
7.5
5.0

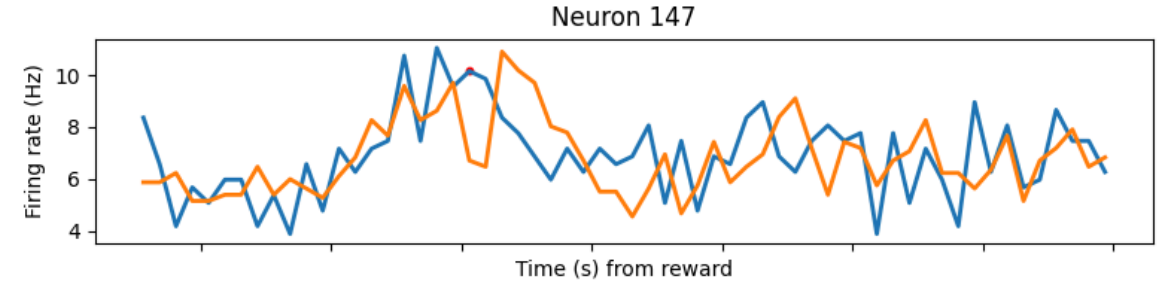

Fi
2.5
Time (s) from reward

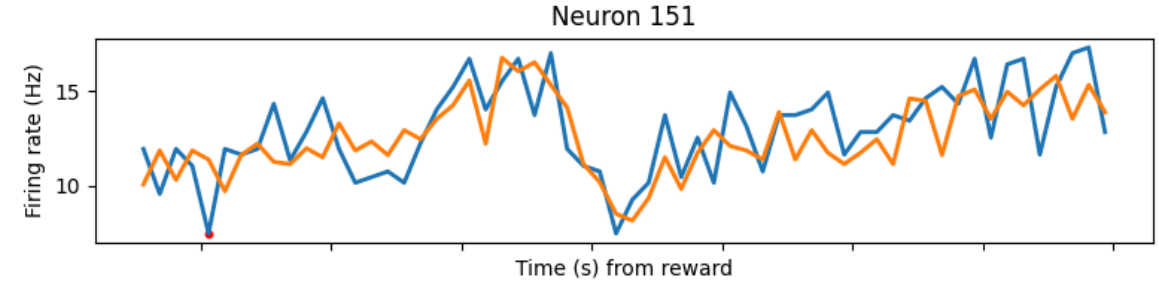

Neuron 153
Firing rate (Hz)
8
6
4
2
Time (s) from reward

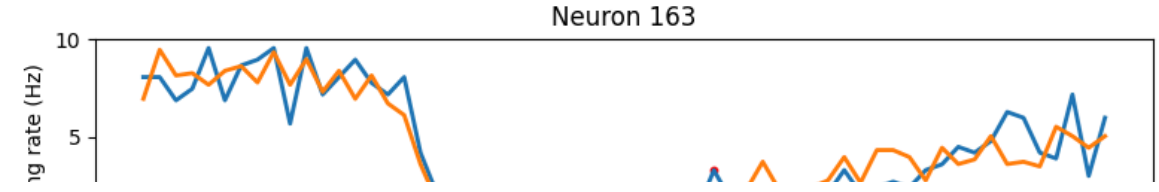

Neuron 154
Firing rate (Hz)
10.0
7.5
5.0
Time (s) from reward

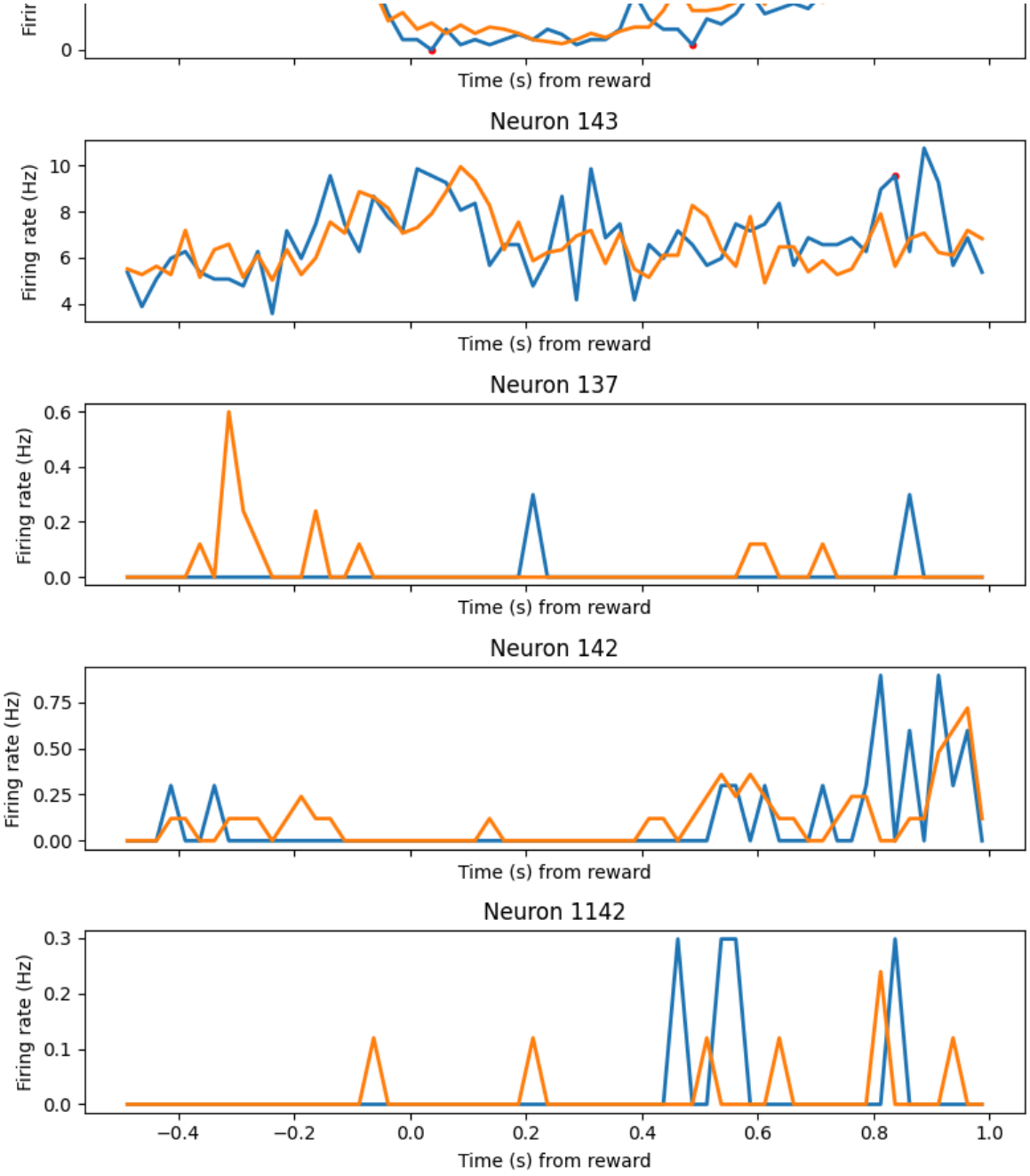

Neuron 1151
Firing rate (Hz)
0.75
0.50
0.25
0.00
Time (s) from reward


Neuron 141
Firing rate (Hz)
8
6
4
Time (s) from reward


Neuron 147
Firing rate (Hz)
10
8
6
4
Time (s) from reward


Neuron 151
Firing rate (Hz)
15
10
Time (s) from reward

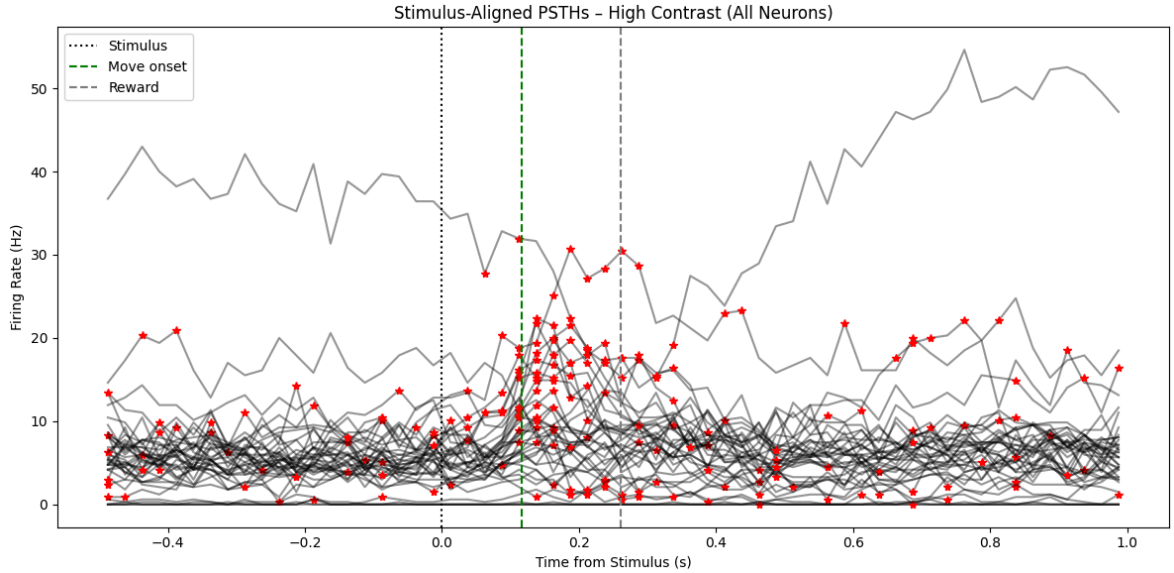

Neuron 163
10
5

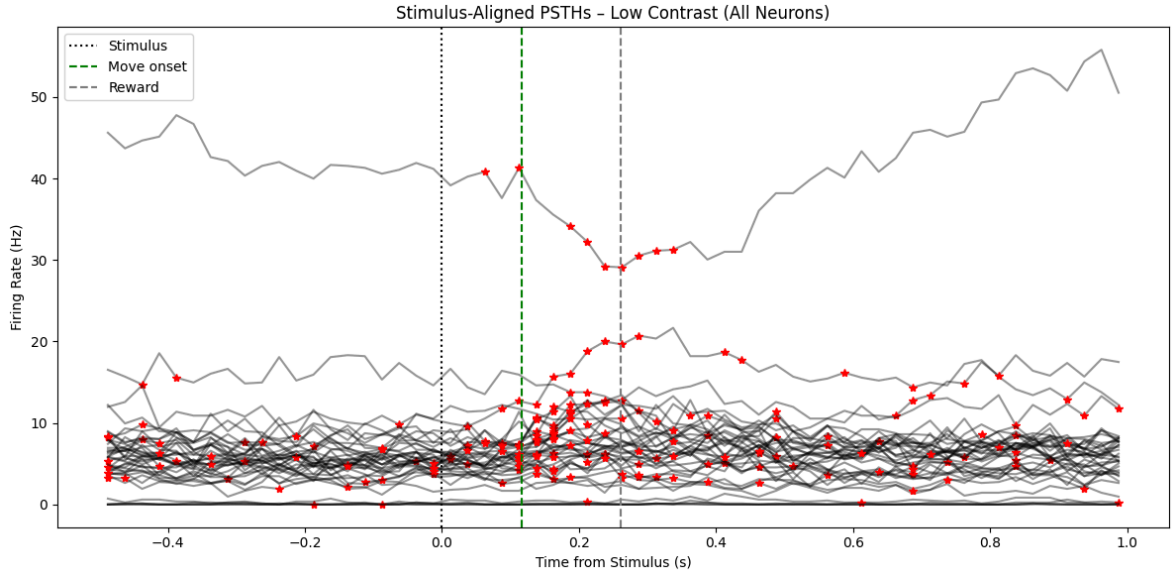


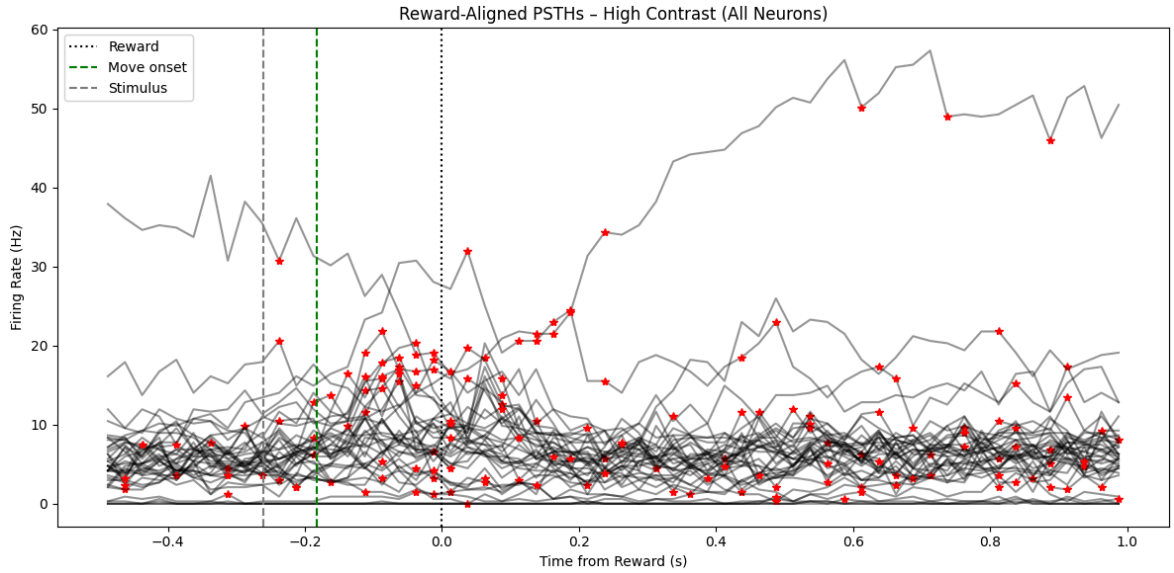
Here, 31 neurons are aligned to the stimulus and reward to see if there are any firing patterns. I separated trials by contrast, computed average firing rates, and performed Mann-Whitney U tests to identify significant differences between different contrast conditions. Significant time bins are marked with red marks.

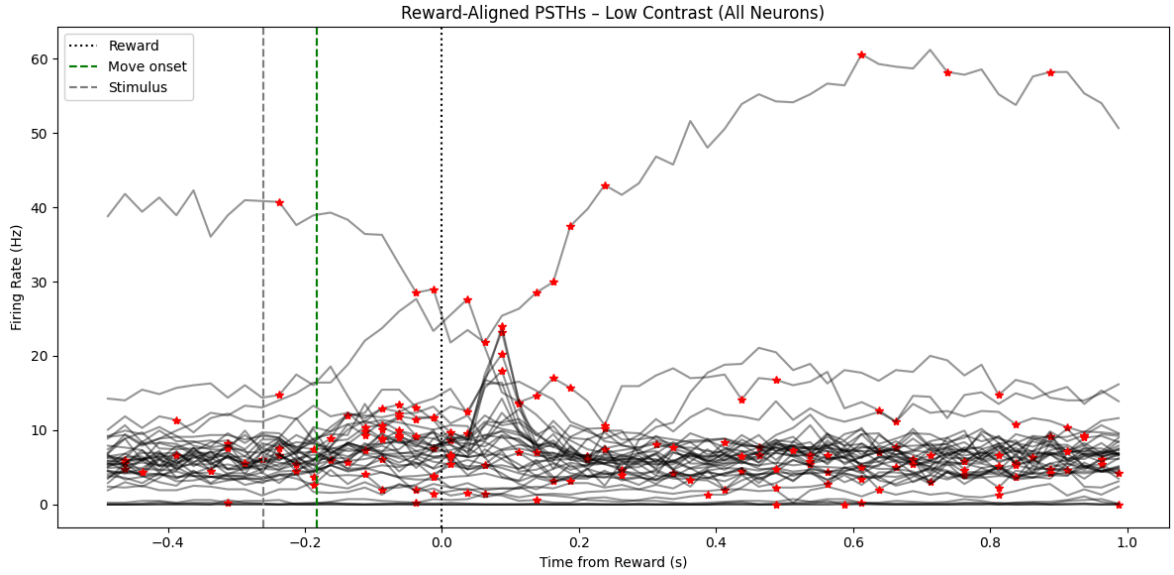
In [33]:
```
def plot_vta_neuron_ids(alignment_times, align_label, contrast_type):
    plt.figure(figsize=(12, 6))

    for neuron_id in vta_neuron_ids:
        spikes_neuron = spikes['times'][spikes['clusters'] == neuron_id]
        high_trials = []
        low_trials = []

        for i, align_t in enumerate(alignment_times):
            aligned = spikes_neuron - align_t
            counts, _ = np.histogram(aligned, bins=T_BINS)

            if contrast_labels[i] == 'high':
```

```
                high_trials.append(counts)
            elif contrast_labels[i] == 'low':
                low_trials.append(counts)

        high_trials = np.array(high_trials)
        low_trials = np.array(low_trials)

        if contrast_type == 'high' and high_trials.shape[0] >= 5:
            fr = high_trials.mean(axis=0) / BIN_SIZE
            plt.plot(TIME_CENTERS, fr, color='black', alpha=0.4)

        elif contrast_type == 'low' and low_trials.shape[0] >= 5:
            fr = low_trials.mean(axis=0) / BIN_SIZE
            plt.plot(TIME_CENTERS, fr, color='black', alpha=0.4)

        # Significance test (high vs low) per bin
        if high_trials.shape[0] >= 5 and low_trials.shape[0] >= 5:
            for b in range(len(TIME_CENTERS)):
                try:
                    _, p = mannwhitneyu(high_trials[:, b], low_trials[:,
                    if p < 0.05:
                        y_val = (high_trials.mean(axis=0)[b] if contrast_
                                 else low_trials.mean(axis=0)[b]) / BIN_S
                        plt.plot(TIME_CENTERS[b], y_val, marker='*', colo
                except:
                    continue

    # Reference lines
    plt.axvline(0, linestyle=':', color='black', label=align_label)
    if align_label == 'Stimulus':
        plt.axvline(move_latency_from_stim, linestyle='--', color='green'
        plt.axvline(reward_latency_from_stim, linestyle='--', color='gray
    else:
        plt.axvline(move_latency_from_reward, linestyle='--', color='gree
        plt.axvline(stim_latency_from_reward, linestyle='--', color='gray

    plt.title(f'{align_label}-Aligned PSTHs — {contrast_type.capitalize()
    plt.xlabel(f'Time from {align_label} (s)')
    plt.ylabel('Firing Rate (Hz)')
    plt.legend()
    plt.tight_layout()
    plt.show()

# Stimulus-aligned, High contrast
plot_vta_neuron_ids(stim_times, align_label='Stimulus', contrast_type='hi
# Stimulus-aligned, Low contrast
plot_vta_neuron_ids(stim_times, align_label='Stimulus', contrast_type='lo
# Reward-aligned, High contrast
plot_vta_neuron_ids(reward_times, align_label='Reward', contrast_type='hi
# Reward-aligned, Low contrast
plot_vta_neuron_ids(reward_times, align_label='Reward', contrast_type='lo
```


Stimulus-Aligned PSTHs – High Contrast (All Neurons)
Stimulus
Move onset
Reward
Firing Rate (Hz)
Time from Stimulus (s)


Stimulus-Aligned PSTHs – Low Contrast (All Neurons)
Stimulus
Move onset
Reward
Firing Rate (Hz)
Time from Stimulus (s)

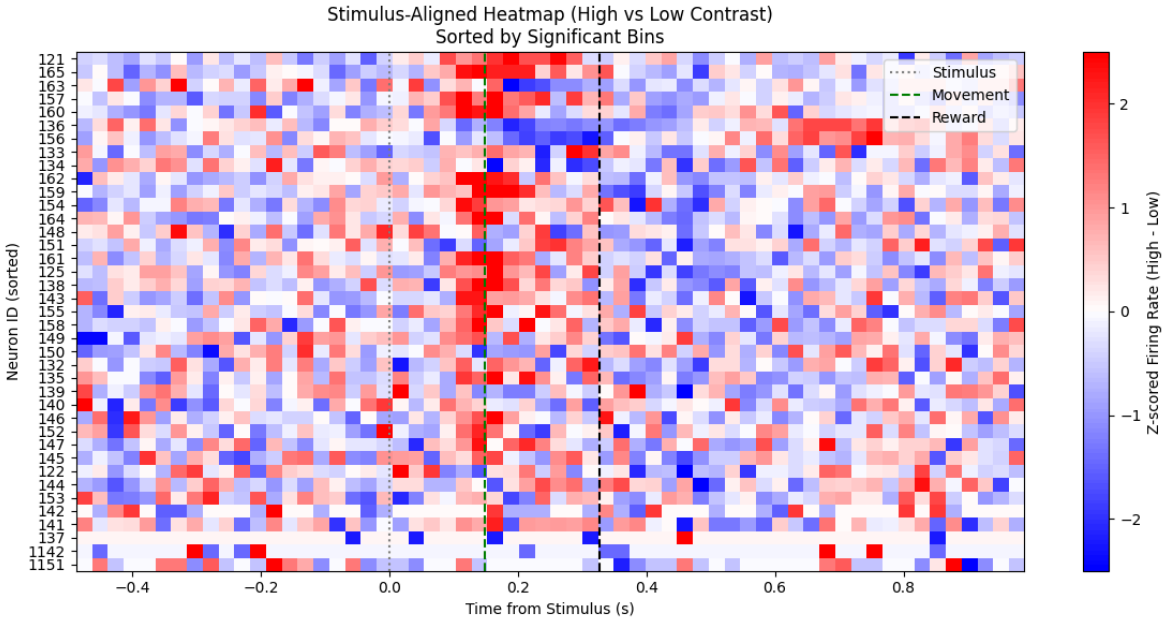

Reward-Aligned PSTHs – High Contrast (All Neurons)
Reward
Move onset
Stimulus
Firing Rate (Hz)
Time from Reward (s)



In high contrast trials aligned to stimulus onset, neural activity shows multiple distinct peaks, particularly around the time of movement initiation. In low contrast trials, this trend appears more diffuse and less sharply defined. When aligned to reward delivery, high contrast trials exhibit numerous significant increases in activity between movement onset and reward delivery, suggesting anticipatory processing. In contrast, low contrast trials show reduced pre-reward activity and more modest peaks following reward, possibly reflecting unexpected reward signaling. Note: There are 3 flat lines in both conditions.

In [32]:
```
#In this section I create stimulus aligned and reward aligned heatmaps.
# Trial selection
correct = trials['feedbackType'] == 1
stim_times = trials['stimOn_times'][correct]
reward_times = trials['feedback_times'][correct]
move_times = trials['firstMovement_times'][correct]
contrast = trials['contrastLeft'].copy()
contrast[np.isnan(contrast)] = trials['contrastRight'][np.isnan(contrast)
contrast = contrast[correct]
contrast_labels = np.where(contrast == 1.0, 'high', 'low')
#reference
mov_on_latency = np.nanmedian(move_times - stim_times)
reward_latency = np.nanmedian(reward_times - stim_times)


# Firing rate matrix and stats

z_matrix = []
sig_bin_counts = []

for nid in vta_neuron_ids:
    spikes_n = spikes['times'][spikes['clusters'] == nid]
    high_mat, low_mat = [], []

    for i, t0 in enumerate(stim_times):
        aligned = spikes_n - t0
        counts, _ = np.histogram(aligned, bins=T_BINS)
        if contrast_labels[i] == 'high':
            high_mat.append(counts)
```

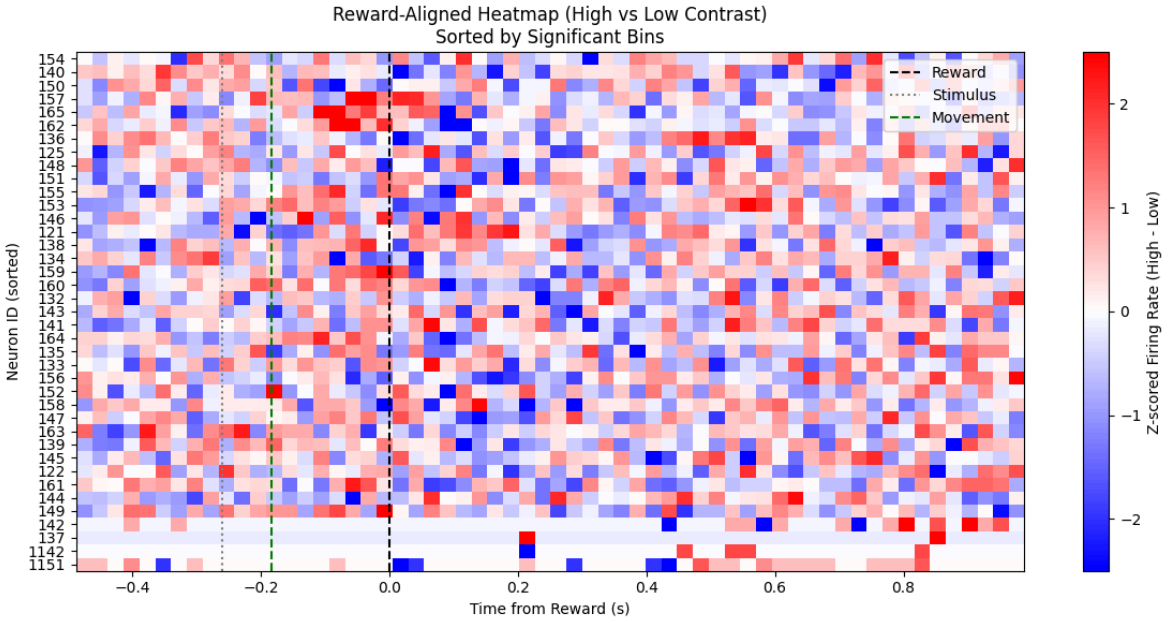
```
        else:
            low_mat.append(counts)

    high_mat = np.array(high_mat)
    low_mat = np.array(low_mat)

    # Compute mean PSTHs
    fr_high = high_mat.mean(axis=0) if len(high_mat) > 0 else np.zeros(le
    fr_low = low_mat.mean(axis=0) if len(low_mat) > 0 else np.zeros(len(T
    fr_diff = fr_high - fr_low

    # Z-score the difference PSTH
    z_diff = zscore(fr_diff)
    z_matrix.append(z_diff)

    # Mann-Whitney test per bin
    sig_count = 0
    for b in range(len(TIME_CENTERS)):
        try:
            if len(high_mat) > 1 and len(low_mat) > 1:
                _, p = mannwhitneyu(high_mat[:, b], low_mat[:, b])
                if p < 0.05:
                    sig_count += 1
        except:
            continue

    sig_bin_counts.append(sig_count)

# Sort by significance (most significant first)
sort_idx = np.argsort(-np.array(sig_bin_counts))
sorted_ids = np.array(vta_neuron_ids)[sort_idx]
sorted_z = np.array(z_matrix)[sort_idx]


# Plot heatmap
plt.figure(figsize=(12, 6))
im = plt.imshow(sorted_z, aspect='auto', cmap='bwr',
                extent=[TIME_CENTERS[0], TIME_CENTERS[-1], len(sorted_ids
                vmin=-2.5, vmax=2.5)

plt.colorbar(im, label='Z-scored Firing Rate (High - Low)')
plt.axvline(0, linestyle=':', color='gray', label='Stimulus')
plt.axvline(mov_on_latency, linestyle='--', color='green', label='Movemen
plt.axvline(reward_latency, linestyle='--', color='black', label='Reward'

plt.yticks(np.arange(len(sorted_ids)) + 0.5, sorted_ids)
plt.xlabel('Time from Stimulus (s)')
plt.ylabel('Neuron ID (sorted)')
plt.title('Stimulus-Aligned Heatmap (High vs Low Contrast)\nSorted by Sig
plt.legend()
plt.tight_layout()
plt.show()
```



In [16]:
```
#Reward aligned heatmap.

# Only 1.0 is high, 0.0 is low contrast
contrast_labels = np.where(contrast == 1.0, 'high',
                    np.where(contrast == 0.0, 'low', 'other'))

stim_times = stim_times[contrast_labels != 'other']
reward_times = reward_times[contrast_labels != 'other']
move_times = move_times[contrast_labels != 'other']
contrast_labels = contrast_labels[contrast_labels != 'other']

# Event latencies
stim_latency = np.nanmedian(stim_times - reward_times)
mov_on_latency = np.nanmedian(move_times - reward_times)

# Firing rate matrix and stats

z_matrix = []
sig_bin_counts = []

for nid in vta_neuron_ids:
    spikes_n = spikes['times'][spikes['clusters'] == nid]
    high_mat, low_mat = [], []

    for i, t0 in enumerate(reward_times):
        aligned = spikes_n - t0
        counts, _ = np.histogram(aligned, bins=T_BINS)
        if contrast_labels[i] == 'high':
            high_mat.append(counts)
        elif contrast_labels[i] == 'low':
            low_mat.append(counts)

    high_mat = np.array(high_mat)
    low_mat = np.array(low_mat)

    fr_high = high_mat.mean(axis=0) if len(high_mat) > 0 else np.zeros(le
    fr_low = low_mat.mean(axis=0) if len(low_mat) > 0 else np.zeros(len(T
    fr_diff = fr_high - fr_low

    z_diff = zscore(fr_diff)
```

```
    z_matrix.append(z_diff)

    sig_count = 0
    for b in range(len(TIME_CENTERS)):
        try:
            if len(high_mat) > 1 and len(low_mat) > 1:
                _, p = mannwhitneyu(high_mat[:, b], low_mat[:, b])
                if p < 0.05:
                    sig_count += 1
        except:
            continue

    sig_bin_counts.append(sig_count)

# Sort by significance (most significant first)
sort_idx = np.argsort(-np.array(sig_bin_counts))
sorted_ids = np.array(vta_neuron_ids)[sort_idx]
sorted_z = np.array(z_matrix)[sort_idx]

# Determine significant bin positions
sig_marks = []
for i, idx in enumerate(sort_idx):
    high_mat = []
    low_mat = []
    spikes_n = spikes['times'][spikes['clusters'] == vta_neuron_ids[idx]]
    for j, t0 in enumerate(reward_times):
        aligned = spikes_n - t0
        counts, _ = np.histogram(aligned, bins=T_BINS)
        if contrast_labels[j] == 'high':
            high_mat.append(counts)
        elif contrast_labels[j] == 'low':
            low_mat.append(counts)
    high_mat = np.array(high_mat)
    low_mat = np.array(low_mat)
    row_sig_bins = []
    for b in range(len(TIME_CENTERS)):
        try:
            if len(high_mat) > 1 and len(low_mat) > 1:
                _, p = mannwhitneyu(high_mat[:, b], low_mat[:, b])
                if p < 0.05:
                    row_sig_bins.append(b)
        except:
            continue
    for b in row_sig_bins:
        sig_marks.append((i, b))  # row, column


# Plot heatmap

plt.figure(figsize=(12, 6))
im = plt.imshow(sorted_z, aspect='auto', cmap='bwr',
                extent=[TIME_CENTERS[0], TIME_CENTERS[-1], len(sorted_ids
                vmin=-2.5, vmax=2.5)


plt.colorbar(im, label='Z-scored Firing Rate (High - Low)')
plt.axvline(0, linestyle='--', color='black', label='Reward')
plt.axvline(stim_latency, linestyle=':', color='gray', label='Stimulus')
plt.axvline(mov_on_latency, linestyle='--', color='green', label='Movemen
```

```
plt.yticks(np.arange(len(sorted_ids)) + 0.5, sorted_ids)
plt.xlabel('Time from Reward (s)')
plt.ylabel('Neuron ID (sorted)')
plt.title('Reward-Aligned Heatmap (High vs Low Contrast)\nSorted by Signi
plt.legend()
plt.tight_layout()
plt.show()
```



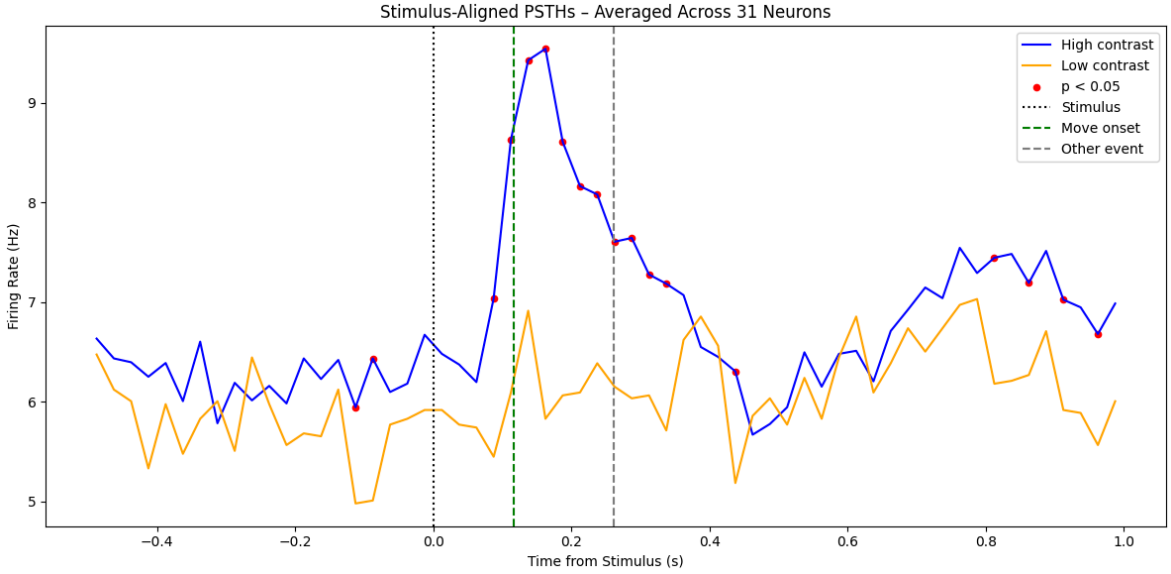
In both reward and stimulus aligned heatmaps neuon: 142, 1142 and 1151 is at the bottom. These are the flat lines previously noted in 2.7.

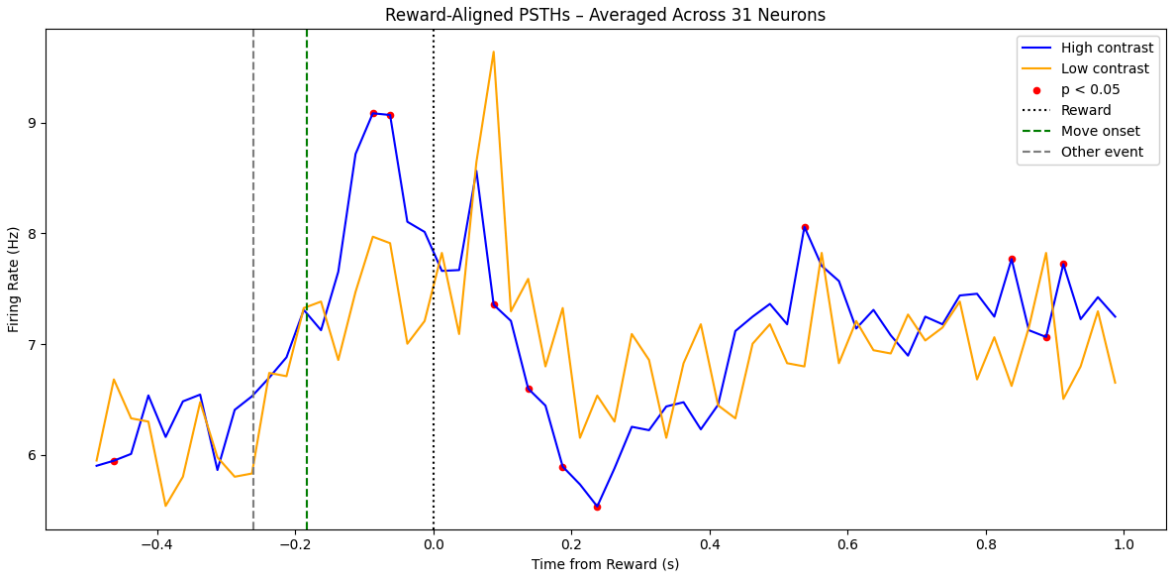
This part of the script averages population-level firing rate responses across 31 VTA neurons, comparing high- vs low-contrast trials aligned to either stimulus or reward onset. For each alignment, it computes per-bin mean firing rates and performs Mann-Whitney U tests across contrast conditions to identify time bins with significant differences. The significant points are marked on the average plots and vertical lines denoting median latencies to movement and reward or stimulus.

In [25]:
```
pre_time = -0.5
post_time = 1.0

# Trial Selection
correct = trials['feedbackType'] == 1
stim_times = trials['stimOn_times'][correct]
reward_times = trials['feedback_times'][correct]
move_times = trials['firstMovement_times'][correct]

# Combine contrasts
contrast = trials['contrastLeft'].copy()
contrast[np.isnan(contrast)] = trials['contrastRight'][np.isnan(contrast)
contrast = contrast[correct]

# Keep only 0.0 and 1.0 contrasts for analysis
contrast_labels = np.where(contrast == 1.0, 'high',
                    np.where(contrast == 0.0, 'low', 'other'))
```

```
# Apply the same filtering to stim_times, reward_times, and move_times
stim_times = stim_times[contrast_labels != 'other']
reward_times = reward_times[contrast_labels != 'other']
move_times = move_times[contrast_labels != 'other']
contrast_labels = contrast_labels[contrast_labels != 'other']


def compute_mean_psth(alignment_times):
    all_high_trials = []
    all_low_trials = []

    for neuron_id in vta_neuron_ids:
        spikes_neuron = spikes['times'][spikes['clusters'] == neuron_id]
        high_matrix, low_matrix = [], []

        for i, align_t in enumerate(alignment_times):
            aligned = spikes_neuron - align_t
            counts, _ = np.histogram(aligned, bins=T_BINS)

            if contrast_labels[i] == 'high':
                high_matrix.append(counts)
            elif contrast_labels[i] == 'low':
                low_matrix.append(counts)

        if len(high_matrix) >= 5:
            all_high_trials.append(np.array(high_matrix))
        if len(low_matrix) >= 5:
            all_low_trials.append(np.array(low_matrix))

    all_high_trials = np.concatenate(all_high_trials, axis=0)
    all_low_trials = np.concatenate(all_low_trials, axis=0)

    mean_high = np.mean(all_high_trials, axis=0) / BIN_SIZE
    mean_low = np.mean(all_low_trials, axis=0) / BIN_SIZE

    # Mann-whitney
    pvals = []
    for b in range(len(TIME_CENTERS)):
        try:
            _, p = mannwhitneyu(all_high_trials[:, b], all_low_trials[:,
        except:
            p = 1.0
        pvals.append(p)
    sig_bins = np.array(pvals) < 0.05

    return mean_high, mean_low, sig_bins

# Plot function
def plot_avg_psth(mean_high, mean_low, sig_bins, align_label, move_lat, e
    plt.figure(figsize=(12, 6))
    plt.plot(TIME_CENTERS, mean_high, color='blue', label='High contrast'
    plt.plot(TIME_CENTERS, mean_low, color='orange', label='Low contrast'

    # Plot significance markers on high contrast line
    plt.scatter(TIME_CENTERS[sig_bins], mean_high[sig_bins], color='red',

    # time points
    plt.axvline(0, linestyle=':', color='black', label=align_label)
    plt.axvline(move_lat, linestyle='--', color='green', label='Move onse
    plt.axvline(event_lat, linestyle='--', color='gray', label='Other eve
```

```
    plt.title(f'{align_label}-Aligned PSTHs — Averaged Across 31 Neurons'
    plt.xlabel(f'Time from {align_label} (s)')
    plt.ylabel('Firing Rate (Hz)')
    plt.legend()
    plt.tight_layout()
    plt.show()


move_latency_from_stim = np.nanmedian(move_times - stim_times)
reward_latency_from_stim = np.nanmedian(reward_times - stim_times)
move_latency_from_reward = np.nanmedian(move_times - reward_times)
stim_latency_from_reward = np.nanmedian(stim_times - reward_times)


# Stimulus-aligned plot
mean_high, mean_low, sig_bins = compute_mean_psth(stim_times)
plot_avg_psth(mean_high, mean_low, sig_bins, align_label='Stimulus',
              move_lat=move_latency_from_stim, event_lat=reward_latency_f

# Reward-aligned plot
mean_high, mean_low, sig_bins = compute_mean_psth(reward_times)
plot_avg_psth(mean_high, mean_low, sig_bins, align_label='Reward',
              move_lat=move_latency_from_reward, event_lat=stim_latency_f
```





This is another way to see the results, more minimalistic. These are population level responses. Again, when we aligned the data to the stimulus the high-contrast activity peaks at the movement. When we align the data to the reward high-contrast trials have 2 visible peaks big one between the movement and the reward (anticipatory signal) followed by a smaller peak after the reward (signaling expected reward). The low contrast condition also has 2 peaks a small one again between the movement onset and the reward onset (?) followed by a bigger peak after the reward (unexpected reward signaling). These plots are supporting my hypothesis.

In this section we compute statistical tests for each selected neuron and list them all. For every bin, it computes the p-value (Mann-Whitney U test), direction of modulation (high > low or vice versa), and effect size (Cohen's d). Significant results ($p < 0.05$) are split into pre-reward and post-reward phases and summarized in tables. This allows detection of temporally precise, contrast-dependent modulations in VTA neuron activity surrounding reward delivery.

In [26]:
```
# Per-Bin Significance & Table

results = []

for nid in vta_neuron_ids:
    spikes_n = spikes['times'][spikes['clusters'] == nid]
    high_mat, low_mat = [], []

    for i, t0 in enumerate(reward_times):
        aligned = spikes_n - t0
        counts, _ = np.histogram(aligned, bins=T_BINS)
        if contrast_labels[i] == 'high':
            high_mat.append(counts)
        elif contrast_labels[i] == 'low':
            low_mat.append(counts)

    high_mat = np.array(high_mat)
    low_mat = np.array(low_mat)

    for b, t in enumerate(TIME_CENTERS):
        try:
            if high_mat.shape[0] > 1 and low_mat.shape[0] > 1:
                _, p = mannwhitneyu(high_mat[:, b], low_mat[:, b])
                diff = high_mat[:, b].mean() - low_mat[:, b].mean()
                direction = 'high > low' if diff > 0 else 'low > high'
                pooled_std = np.sqrt((np.var(high_mat[:, b]) + np.var(low
                cohen_d = diff / pooled_std if pooled_std > 0 else 0.0
                if p < 0.05:
                    results.append({
                        'neuron': nid,
                        'time': round(t, 3),
                        'p_value': round(p, 6),
                        'cohen_d': round(cohen_d, 4),
                        'direction': direction,
                        'phase': 'pre-reward' if t < 0 else 'post-reward'
                    })
        except:
```

```
            continue


# Summary Tables
sig_df = pd.DataFrame(results)
sig_df = sig_df.round({'time': 3, 'p_value': 6, 'cohen_d': 4})

pre_reward = sig_df[sig_df['phase'] == 'pre-reward'].sort_values(by=['neu
post_reward = sig_df[sig_df['phase'] == 'post-reward'].sort_values(by=['n

from tabulate import tabulate

print("\n PRE-REWARD SIGNIFICANT TIME BINS:\n")
print(tabulate(pre_reward[['neuron', 'time', 'p_value', 'cohen_d', 'direc
               headers='keys', tablefmt='fancy_grid', showindex=False))

print("\n POST-REWARD SIGNIFICANT TIME BINS:\n")
print(tabulate(post_reward[['neuron', 'time', 'p_value', 'cohen_d', 'dire
               headers='keys', tablefmt='fancy_grid', showindex=False))
```

PRE-REWARD SIGNIFICANT TIME BINS:

neuron	time	p_value	cohen_d	direction
121	-0.237	0.00424	-0.4786	low > high
125	-0.462	0.000997	-0.4705	low > high
133	-0.412	0.035728	-0.3146	low > high
134	-0.287	0.046948	0.483	high > low
134	-0.087	0.015014	-0.4129	low > high
135	-0.187	0.039905	-0.3611	low > high
135	-0.012	0.030865	-0.3333	low > high
136	-0.037	0.008083	-0.379	low > high
138	-0.387	0.002866	-0.5125	low > high
140	-0.362	0.017071	0.532	high > low
146	-0.287	0.04645	-0.3141	low > high
146	-0.212	0.002233	-0.4468	low > high
146	-0.137	0.006573	0.6243	high > low
148	-0.462	0.010499	-0.2925	low > high
150	-0.112	0.01788	-0.3912	low > high
150	-0.087	0.01678	-0.4074	low > high
150	-0.012	0.00833	-0.4107	low > high
151	-0.462	0.042363	-0.3392	low > high
152	-0.187	0.003928	0.6961	high > low
154	-0.337	0.02325	-0.3824	low > high
154	-0.312	0.031589	0.4908	high > low
155	-0.387	0.000606	-0.5712	low > high
155	-0.312	0.024005	-0.3872	low > high
155	-0.087	0.037092	0.3863	high > low
157	-0.212	0.009682	0.5895	high > low
157	-0.062	0.005736	0.6009	high > low
157	-0.037	0.011515	0.5719	high > low
159	-0.412	0.03105	0.4891	high > low

159	-0.012	0.007296	0.5473	high > low
162	-0.087	0.014256	0.4812	high > low
162	-0.062	0.01994	0.5125	high > low
162	-0.012	0.037808	0.4576	high > low
165	-0.337	0.004539	-0.5076	low > high
165	-0.112	0.000967	0.7267	high > low
165	-0.087	0.004958	0.6284	high > low
165	-0.037	0.012436	0.5501	high > low

POST-REWARD SIGNIFICANT TIME BINS:

neuron	time	p_value	cohen_d	direction
121	0.113	0.034046	0.4459	high > low
121	0.163	0.020544	0.4637	high > low
121	0.188	0.013383	0.512	high > low
122	0.138	0.035386	-0.35	low > high
125	0.263	0.036505	-0.3621	low > high
125	0.288	0.009717	-0.4166	low > high
125	0.488	0.030865	-0.3333	low > high
125	0.963	0.035149	-0.3615	low > high
132	0.238	0.007699	-0.454	low > high
132	0.338	0.018571	0.5748	high > low
132	0.538	0.031232	0.3675	high > low
133	0.513	0.036505	-0.3621	low > high
134	0.038	0.045078	-0.3437	low > high
134	0.938	0.024005	-0.3872	low > high
135	0.388	0.012049	-0.4215	low > high
136	0.013	0.004068	-0.5137	low > high
136	0.138	0.000445	-0.5951	low > high
136	0.238	0.010302	-0.4858	low > high
136	0.288	0.002538	-0.3272	low > high

138	0.288	0.011004	−0.4627	low > high
138	0.563	0.019375	−0.4197	low > high
138	0.763	0.018792	−0.456	low > high
139	0.113	0.007052	−0.4871	low > high
139	0.888	0.005859	−0.4538	low > high
140	0.013	0.001632	−0.5398	low > high
140	0.238	0.023329	−0.3652	low > high
140	0.613	0.049046	−0.3748	low > high
140	0.713	0.020619	−0.4019	low > high
140	0.813	0.035919	0.4685	high > low
141	0.063	0.031235	0.4883	high > low
141	0.313	0.016269	−0.3764	low > high
141	0.663	0.047049	0.4596	high > low
143	0.288	0.01078	−0.4482	low > high
143	0.688	0.037866	0.4706	high > low
143	0.863	0.016305	−0.3557	low > high
144	0.263	0.02602	0.5067	high > low
145	0.413	0.004943	−0.442	low > high
145	0.563	0.030973	−0.3962	low > high
146	0.063	0.013424	−0.3902	low > high
147	0.138	0.005173	−0.5034	low > high
147	0.788	0.043131	0.4518	high > low
148	0.113	0.027083	0.5151	high > low
148	0.188	0.003563	−0.5257	low > high
148	0.988	0.035919	0.4685	high > low
149	0.963	0.037243	−0.3309	low > high
150	0.188	0.033002	−0.346	low > high
150	0.563	0.045078	−0.3437	low > high
150	0.688	0.005329	−0.4165	low > high
151	0.188	0.001044	−0.6262	low > high

151	0.388	0.016564	−0.4277	low > high
151	0.463	0.032148	−0.3997	low > high
152	0.088	0.008714	−0.4255	low > high
153	0.088	0.008973	−0.3586	low > high
153	0.363	0.00833	−0.4107	low > high
153	0.413	0.037243	−0.3309	low > high
153	0.788	0.010535	−0.327	low > high
154	0.113	0.03105	0.4891	high > low
154	0.313	0.000132	−0.6351	low > high
154	0.638	0.031235	0.4883	high > low
154	0.763	0.030536	0.4788	high > low
154	0.813	0.02932	−0.3148	low > high
155	0.088	0.015187	−0.5377	low > high
156	0.138	0.018058	−0.2458	low > high
156	0.163	0.048324	−0.311	low > high
157	0.013	0.004312	0.6524	high > low
157	0.038	0.031268	0.486	high > low
157	0.888	0.014856	−0.3825	low > high
158	0.063	0.049112	−0.2838	low > high
158	0.288	0.005691	−0.3482	low > high
159	0.713	0.043641	0.4475	high > low
159	0.938	0.024939	−0.3697	low > high
160	0.088	0.018621	−0.4389	low > high
160	0.363	0.043641	0.4475	high > low
160	0.663	0.037172	0.4586	high > low
161	0.288	0.040952	−0.324	low > high
162	0.088	0.002407	−0.5688	low > high
162	0.113	0.013045	−0.482	low > high
162	0.813	0.038161	−0.3212	low > high
163	0.088	0.049112	−0.2838	low > high

163	0.663	0.041579	-0.2164	low > high
164	0.088	6.2e-05	-0.6863	low > high
164	0.813	0.044739	-0.3972	low > high
164	0.838	0.016883	0.5344	high > low
165	0.388	0.011699	-0.467	low > high
165	0.913	0.011506	0.5657	high > low

In [27]:
```
# Remove invalid trials
valid = contrast_labels != 'other'
reward_times = reward_times[valid]
contrast_labels = contrast_labels[valid]

# Analysis
summary_rows = []

for nid in vta_neuron_ids:
    spikes_n = spikes['times'][spikes['clusters'] == nid]
    high_mat, low_mat = [], []

    for i, t0 in enumerate(reward_times):
        aligned = spikes_n - t0
        counts, _ = np.histogram(aligned, bins=T_BINS)
        if contrast_labels[i] == 'high':
            high_mat.append(counts)
        elif contrast_labels[i] == 'low':
            low_mat.append(counts)

    high_mat = np.array(high_mat)
    low_mat = np.array(low_mat)

    pre_pvals, pre_dir = [], []
    post_pvals, post_dir = [], []

    for b, t in enumerate(TIME_CENTERS):
        if len(high_mat) > 1 and len(low_mat) > 1:
            try:
                _, p = mannwhitneyu(high_mat[:, b], low_mat[:, b], altern
                if p < 0.05:
                    mean_h = high_mat[:, b].mean()
                    mean_l = low_mat[:, b].mean()
                    direction = 'high > low' if mean_h > mean_l else 'low
                    if t < 0:
                        pre_pvals.append(round(p, 4))
                        pre_dir.append(direction)
                    else:
                        post_pvals.append(round(p, 4))
                        post_dir.append(direction)
            except:
                continue

    # hypothesis match
    match_hypothesis = (
        ('high > low' in pre_dir) and ('low > high' in post_dir)
```

```
    )

    summary_rows.append({
        'neuron': nid,
        'pre_reward_pvals': pre_pvals,
        'pre_direction': ', '.join(pre_dir),
        'post_reward_pvals': post_pvals,
        'post_direction': ', '.join(post_dir),
        'fits_hypothesis': 'TRUE' if match_hypothesis else ''
    })

# Create DataFrame
summary_df = pd.DataFrame(summary_rows)
pd.set_option("display.max_colwidth", None)
display(summary_df)
```

	neuron	pre_reward_pvals	pre_direction	post_reward_pvals	post_direction	fits
0	121	[0.0042]	low > high	[0.034, 0.0205, 0.0134]	high > low, high > low, high > low	
1	122	[]		[0.0354]	low > high	
2	125	[0.001]	low > high	[0.0365, 0.0097, 0.0309, 0.0351]	low > high, low > high, low > high, low > high	
3	132	[]		[0.0077, 0.0186, 0.0312]	low > high, high > low, high > low	
4	133	[0.0357]	low > high	[0.0365]	low > high	
5	134	[0.0469, 0.015]	high > low, low > high	[0.0451, 0.024]	low > high, low > high	
6	135	[0.0399, 0.0309]	low > high, low > high	[0.012]	low > high	
7	136	[0.0081]	low > high	[0.0041, 0.0004, 0.0103, 0.0025]	low > high, low > high, low > high, low > high	
8	137	[]		[]		
9	138	[0.0029]	low > high	[0.011, 0.0194, 0.0188]	low > high, low > high, low > high	
10	139	[]		[0.0071, 0.0059]	low > high, low > high	
11	140	[0.0171]	high > low	[0.0016, 0.0233, 0.049, 0.0206, 0.0359]	low > high, low > high, low > high, low > high, high > low	
12	141	[]		[0.0312, 0.0163, 0.047]	high > low, low > high, high > low	
13	142	[]		[]		
14	143	[]		[0.0108, 0.0379, 0.0163]	low > high, high > low, low > high	
15	144	[]		[0.026]	high > low	
16	145	[]		[0.0049, 0.031]	low > high, low > high	
17	146	[0.0464, 0.0022, 0.0066]	low > high, low > high, high > low	[0.0134]	low > high	
18	147	[]		[0.0052, 0.0431]	low > high, high > low	

	neuron	pre_reward_pvals	pre_direction	post_reward_pvals	post_direction	fits
19	148	[0.0105]	low > high	[0.0271, 0.0036, 0.0359]	high > low, low > high, high > low	
20	149	[]		[0.0372]	low > high	
21	150	[0.0179, 0.0168, 0.0083]	low > high, low > high, low > high	[0.033, 0.0451, 0.0053]	low > high, low > high, low > high	
22	151	[0.0424]	low > high	[0.001, 0.0166, 0.0321]	low > high, low > high, low > high	
23	152	[0.0039]	high > low	[0.0087]	low > high	
24	153	[]		[0.009, 0.0083, 0.0372, 0.0105]	low > high, low > high, low > high, low > high	
25	154	[0.0233, 0.0316]	low > high, high > low	[0.031, 0.0001, 0.0312, 0.0305, 0.0293]	high > low, low > high, high > low, high > low, low > high	
26	155	[0.0006, 0.024, 0.0371]	low > high, low > high, high > low	[0.0152]	low > high	
27	156	[]		[0.0181, 0.0483]	low > high, low > high	
28	157	[0.0097, 0.0057, 0.0115]	high > low, high > low, high > low	[0.0043, 0.0313, 0.0149]	high > low, high > low, low > high	
29	158	[]		[0.0491, 0.0057]	low > high, low > high	
30	159	[0.031, 0.0073]	high > low, high > low	[0.0436, 0.0249]	high > low, low > high	
31	160	[]		[0.0186, 0.0436, 0.0372]	low > high, high > low, high > low	
32	161	[]		[0.041]	low > high	
33	162	[0.0143, 0.0199, 0.0378]	high > low, high > low, high > low	[0.0024, 0.013, 0.0382]	low > high, low > high, low > high	
34	163	[]		[0.0491, 0.0416]	low > high, low > high	
35	164	[]		[0.0001, 0.0447, 0.0169]	low > high, low > high, high > low	
36	165	[0.0045, 0.001, 0.005, 0.0124]	low > high, high > low, high > low, high > low	[0.0117, 0.0115]	low > high, high > low	

	neuron	pre_reward_pvals	pre_direction	post_reward_pvals	post_direction	fits
37	1142	[]		[]		
38	1151	[]		[]		

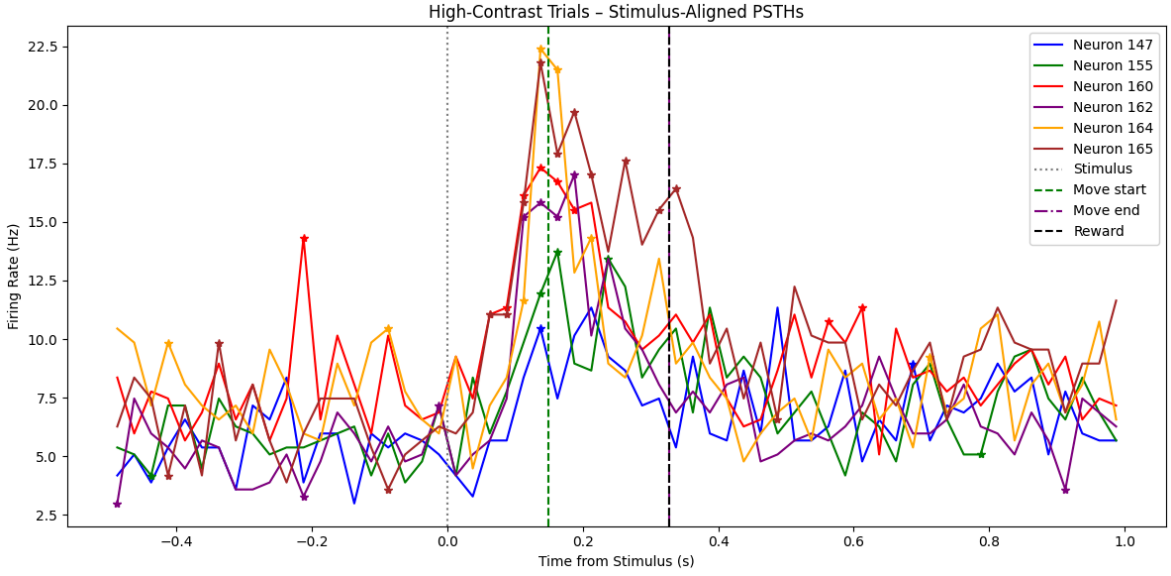
In [29]:
```
# These 6 neurons are picked for further analysis.
#parameters
selected_neurons = [147, 155, 160, 162, 164, 165]

# Correct reward trials
correct_trials = trials['feedbackType'] == 1

# Contrast labels
contrast = trials['contrastLeft'].copy()
contrast[np.isnan(contrast)] = trials['contrastRight'][np.isnan(contrast)
contrast = contrast[correct_trials]
contrast_labels = np.where(contrast == 1.0, 'high', 'low')

# Core timestamps
stim_times    = trials['stimOn_times'][correct_trials]
reward_times  = trials['feedback_times'][correct_trials]

move_on_times = trials['firstMovement_times'][correct_trials]   # movemen
move_off_times = trials['response_times'][correct_trials]       # movemen
mov_on_latency   = np.nanmedian(move_on_times  - stim_times)
mov_off_latency  = np.nanmedian(move_off_times - stim_times)
reward_latency   = np.nanmedian(reward_times   - stim_times)

firing_by_neuron = {}
stat_results     = {}

for neuron_id in selected_neurons:
    spikes_neuron = spikes['times'][spikes['clusters'] == neuron_id]

    high_mat, low_mat = [], []

    # Build trial-by-bin matrices (stimulus-aligned)
    for idx, stim_t in enumerate(stim_times):
        aligned = spikes_neuron - stim_t
        counts, _ = np.histogram(aligned, bins=T_BINS)

        if contrast_labels[idx] == 'high':
            high_mat.append(counts)
        else:
            low_mat.append(counts)

    high_mat = np.asarray(high_mat)
    low_mat  = np.asarray(low_mat)
    firing_by_neuron[neuron_id] = {'high': high_mat, 'low': low_mat}

    # Mann-Whitney U test at every bin
    bin_pvals = []
    for b in range(len(TIME_CENTERS)):
        if len(high_mat) > 1 and len(low_mat) > 1:
            _, p = mannwhitneyu(high_mat[:, b], low_mat[:, b], alternativ
        else:
```

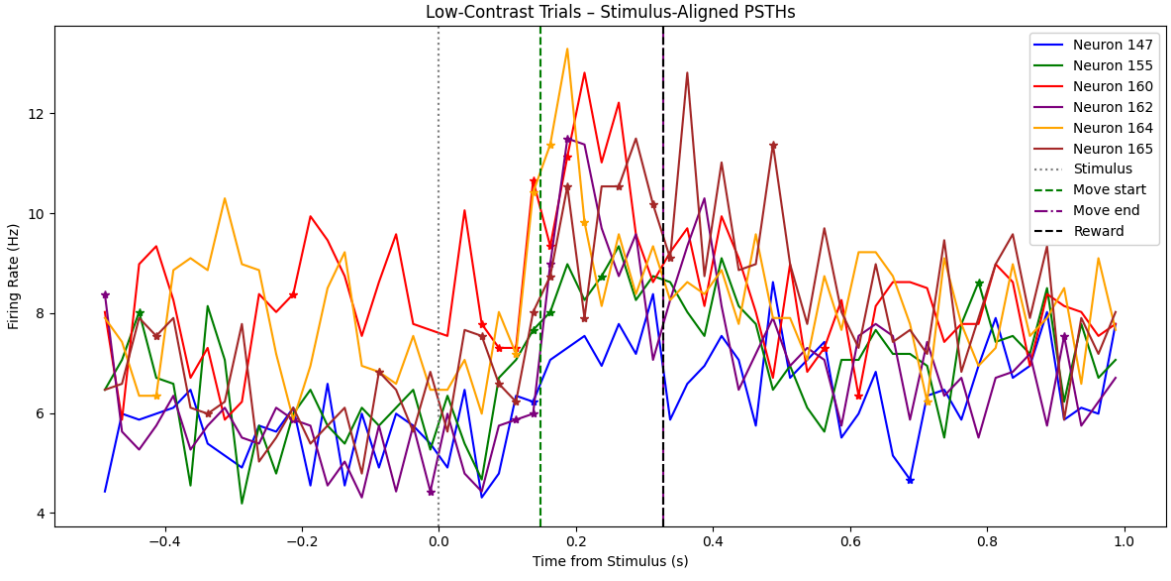
```
            p = 1.0
        bin_pvals.append(p)

    stat_results[neuron_id] = {
        'p_values': bin_pvals,
        'mean_high': high_mat.mean() if high_mat.size else np.nan,
        'mean_low' : low_mat.mean()  if low_mat.size else np.nan,
        'sem_high' : sem(high_mat)   if high_mat.size else np.nan,
        'sem_low'  : sem(low_mat)    if low_mat.size else np.nan
    }

# COMBINED PSTH — HIGH CONTRAST
colors = ['blue','green','red','purple','orange','brown']
plt.figure(figsize=(12,6))

for i, nid in enumerate(selected_neurons):
    fr_high = firing_by_neuron[nid]['high']
    if fr_high.size == 0:                    # skip if no trials
        continue
    mean_high = fr_high.mean(axis=0) / BIN_SIZE
    plt.plot(TIME_CENTERS, mean_high, color=colors[i], label=f'Neuron {ni

    # Mark significant bins
    fr_low = firing_by_neuron[nid]['low']
    for t_idx, t in enumerate(TIME_CENTERS):
        if fr_low.size and len(fr_high) > 5 and len(fr_low) > 5:
            _, p = mannwhitneyu(fr_low[:,t_idx], fr_high[:,t_idx])
            if p < 0.05:
                plt.plot(t, mean_high[t_idx], marker='*', color=colors[i]

# Reference lines
plt.axvline(0,                 ls=':',  color='grey',   label='Stimulus')
plt.axvline(mov_on_latency,    ls='--', color='green',  label='Move start
plt.axvline(mov_off_latency,   ls='-.', color='purple', label='Move end')
plt.axvline(reward_latency,    ls='--', color='black',  label='Reward')

plt.title('High-Contrast Trials — Stimulus-Aligned PSTHs')
plt.xlabel('Time from Stimulus (s)')
plt.ylabel('Firing Rate (Hz)')
plt.legend()
plt.tight_layout()
plt.show()


# LOW CONTRAST

plt.figure(figsize=(12,6))

for i, nid in enumerate(selected_neurons):
    fr_low = firing_by_neuron[nid]['low']
    if fr_low.size == 0:
        continue
    mean_low = fr_low.mean(axis=0) / BIN_SIZE
    plt.plot(TIME_CENTERS, mean_low, color=colors[i], label=f'Neuron {nid

    # Mark significant bins
    fr_high = firing_by_neuron[nid]['high']
    for t_idx, t in enumerate(TIME_CENTERS):
        if fr_high.size and len(fr_high) > 5 and len(fr_low) > 5:
            _, p = mannwhitneyu(fr_low[:,t_idx], fr_high[:,t_idx])
```

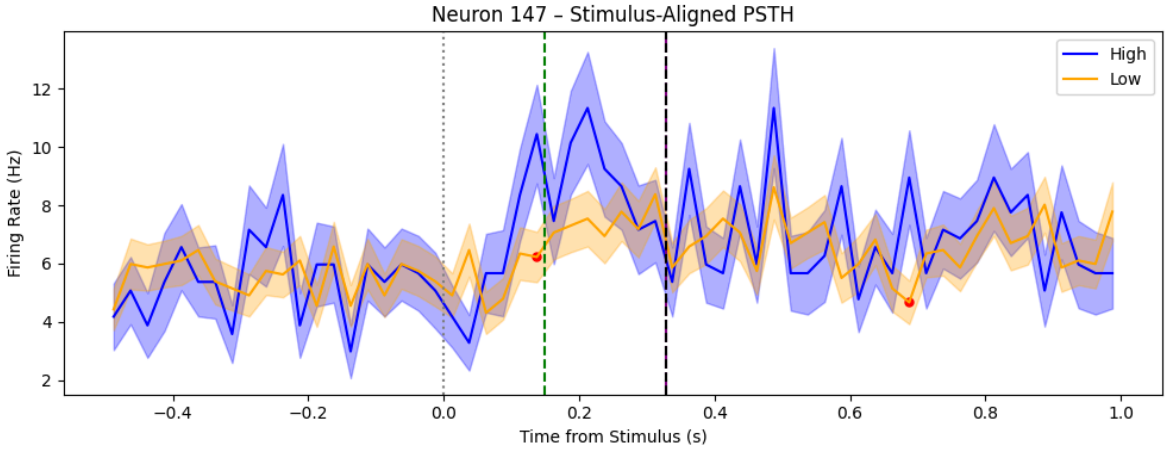
```
            if p < 0.05:
                plt.plot(t, mean_low[t_idx], marker='*', color=colors[i],

# Reference lines
plt.axvline(0,                ls=':',  color='grey',   label='Stimulus')
plt.axvline(mov_on_latency,   ls='--', color='green',  label='Move start
plt.axvline(mov_off_latency,  ls='-.', color='purple', label='Move end')
plt.axvline(reward_latency,   ls='--', color='black',  label='Reward')

plt.title('Low-Contrast Trials — Stimulus-Aligned PSTHs')
plt.xlabel('Time from Stimulus (s)')
plt.ylabel('Firing Rate (Hz)')
plt.legend()
plt.tight_layout()
plt.show()


# INDIVIDUAL NEURON PSTHs WITH SEM & SIG POINTS

for nid in selected_neurons:
    fr_high = firing_by_neuron[nid]['high']
    fr_low  = firing_by_neuron[nid]['low']
    if fr_high.size == 0 or fr_low.size == 0:
        continue

    mean_high = fr_high.mean(axis=0) / BIN_SIZE
    mean_low  = fr_low.mean(axis=0)  / BIN_SIZE
    sem_high  = sem(fr_high, axis=0) / BIN_SIZE
    sem_low   = sem(fr_low,  axis=0) / BIN_SIZE

    plt.figure(figsize=(10,4))
    plt.plot(TIME_CENTERS, mean_high, color='blue',   label='High')
    plt.fill_between(TIME_CENTERS, mean_high - sem_high, mean_high + sem_
                     color='blue', alpha=0.3)
    plt.plot(TIME_CENTERS, mean_low,  color='orange', label='Low')
    plt.fill_between(TIME_CENTERS, mean_low - sem_low,  mean_low + sem_lo
                     color='orange', alpha=0.3)

    # Significant bins (Low vs High)
    sig_idx = np.where(np.array(stat_results[nid]['p_values']) < 0.05)[0]
    plt.scatter(TIME_CENTERS[sig_idx], mean_low[sig_idx], color='red', s=

    # Reference lines
    plt.axvline(0,                ls=':',  color='grey')
    plt.axvline(mov_on_latency,   ls='--', color='green')
    plt.axvline(mov_off_latency,  ls='-.', color='purple')
    plt.axvline(reward_latency,   ls='--', color='black')

    plt.title(f'Neuron {nid} — Stimulus-Aligned PSTH')
    plt.xlabel('Time from Stimulus (s)')
    plt.ylabel('Firing Rate (Hz)')
    plt.legend()
    plt.tight_layout()
    plt.show()

# RASTER PLOTS (High vs Low Contrast)

time_window = (pre_time, post_time)
valid_idxs  = np.where(correct_trials)[0]
```

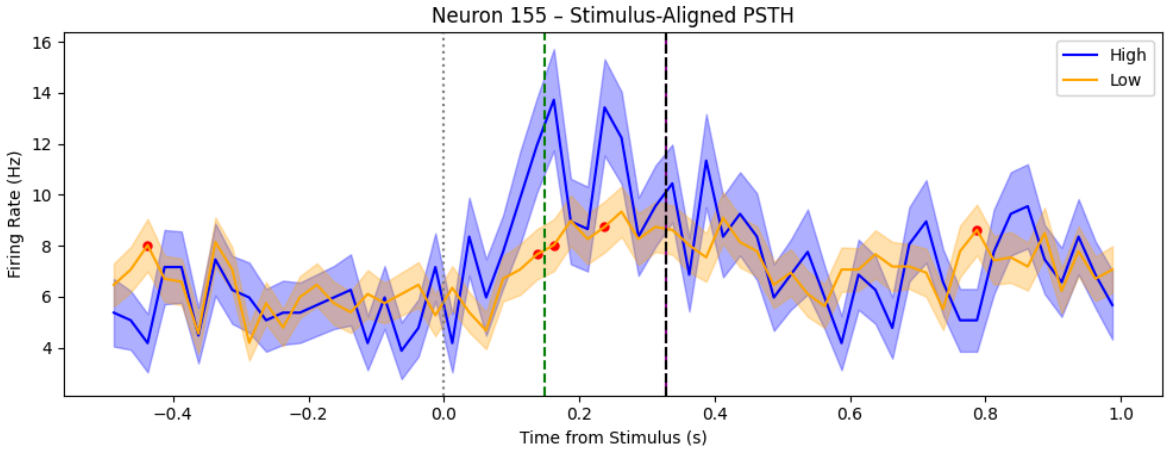
```
for nid in selected_neurons:
    spikes_neuron = spikes['times'][spikes['clusters'] == nid]
    raster = {'high': [], 'low': []}

    for i, trial_idx in enumerate(valid_idxs):
        stim_t = trials['stimOn_times'][trial_idx]
        aligned = spikes_neuron - stim_t
        in_win  = aligned[(aligned >= time_window[0]) & (aligned <= time_
        label   = 'high' if contrast_labels[i] == 'high' else 'low'
        raster[label].append(in_win)

    high_trials = raster['high']
    low_trials  = raster['low']

    plt.figure(figsize=(12,6))
    # High contrast (top)
    for i, spks in enumerate(high_trials):
        plt.vlines(spks, i+0.5, i+1.5, color='black', linewidth=0.5)
    high_end = len(high_trials)

    # Low contrast (bottom)
    for i, spks in enumerate(low_trials):
        plt.vlines(spks, high_end + i + 0.5, high_end + i + 1.5, color='b

    # Reference lines
    plt.axvline(0,                ls=':',  color='grey',   linewidth=1)
    plt.axvline(mov_on_latency,   ls='--', color='green',  linewidth=1)
    plt.axvline(mov_off_latency,  ls='-.', color='purple', linewidth=1)
    plt.axvline(reward_latency,   ls='--', color='black',  linewidth=1)

    plt.ylim(0, high_end + len(low_trials) + 1)
    plt.title(f'Raster — Neuron {nid}\nTop: High | Bottom: Low  (Stimulus
    plt.xlabel('Time from Stimulus (s)')
    plt.ylabel('Trials')
    plt.tight_layout()
    plt.show()

# STATISTICAL SUMMARY TABLE
results = []
all_high, all_low = [], []

for nid in selected_neurons:
    fr_high = firing_by_neuron[nid]['high']
    fr_low  = firing_by_neuron[nid]['low']
    if fr_high.size == 0 or fr_low.size == 0:
        continue

    # Flatten across time & trials for a simple overall comparison
    stat, p_val = mannwhitneyu(fr_high.flatten(), fr_low.flatten(), alter

    m_high = fr_high.mean(); m_low = fr_low.mean()
    pooled_std = np.sqrt((np.var(fr_high, ddof=1) + np.var(fr_low, ddof=1
    cohen_d = (m_high - m_low) / pooled_std if pooled_std > 0 else 0

    results.append({'neuron': nid,
                    'mean_high': round(m_high, 4),
                    'mean_low' : round(m_low,  4),
                    'p_value'  : round(float(p_val), 6),
                    'cohen_d'  : round(cohen_d, 6)})
```

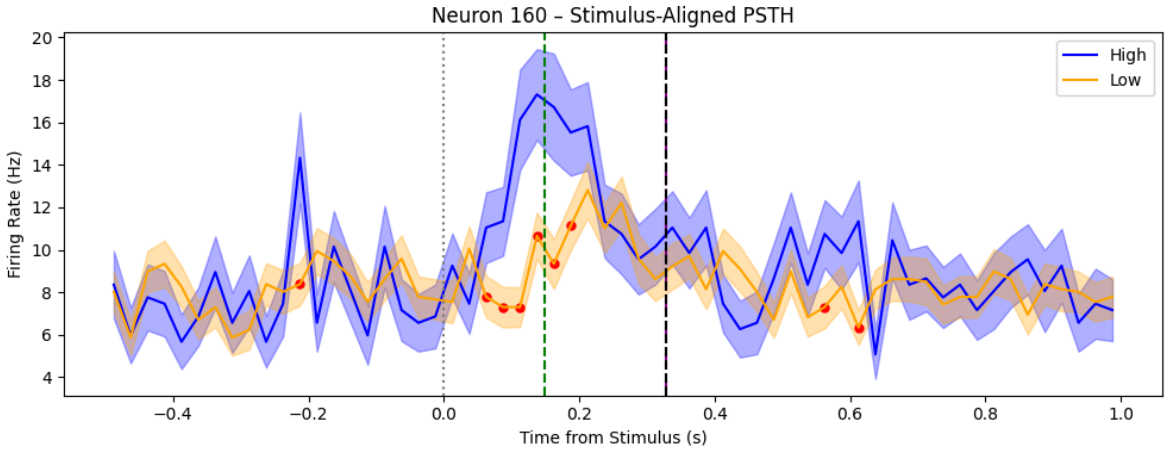
```
    all_high.extend(fr_high.flatten())
    all_low.extend(fr_low.flatten())

# Group-level comparison
group_stat, group_p = mannwhitneyu(all_high, all_low, alternative='two-si
group_mean_high = np.mean(all_high); group_mean_low = np.mean(all_low)
group_pooled_std = np.sqrt((np.var(all_high, ddof=1) + np.var(all_low, dd
group_cohen_d = (group_mean_high - group_mean_low) / group_pooled_std if

group_summary = pd.DataFrame([{
    'mean_high' : round(group_mean_high, 4),
    'mean_low'  : round(group_mean_low,  4),
    'SEM_high'  : round(np.std(all_high, ddof=1) / np.sqrt(len(all_high))
    'SEM_low'   : round(np.std(all_low,  ddof=1) / np.sqrt(len(all_low)),
    'p_value'   : round(float(group_p), 6),
    'cohen_d'   : round(group_cohen_d, 6)
}])

per_neuron_df = pd.DataFrame(results)

print("\nGROUP-LEVEL SUMMARY (Stimulus-Aligned)")
print(group_summary.to_string(index=False))

print("\nPER-NEURON STATS ")
print(per_neuron_df.to_string(index=False))
```

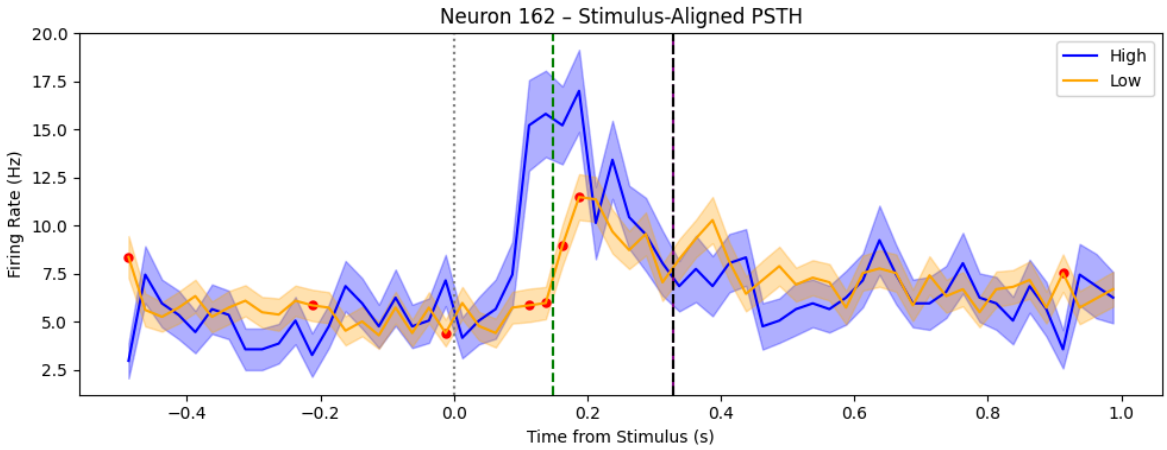


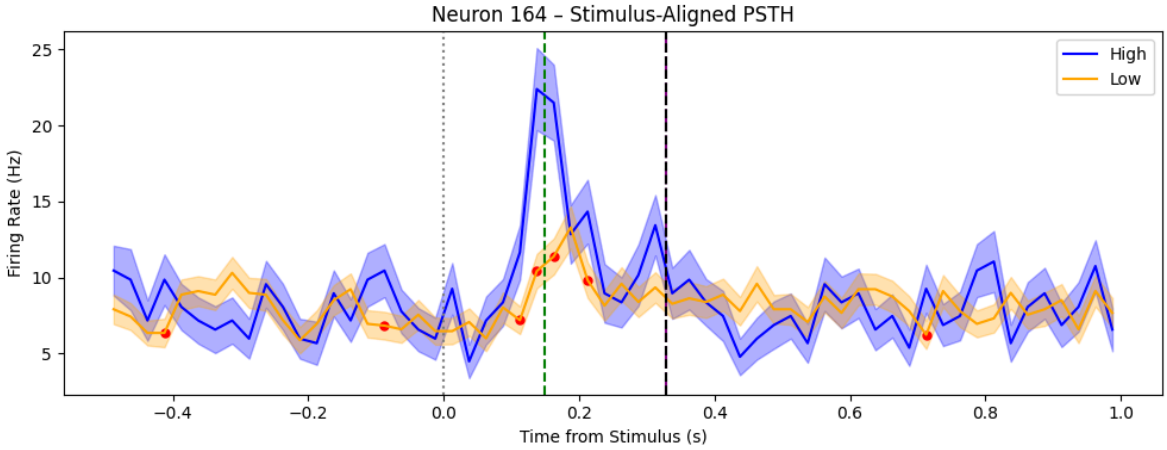


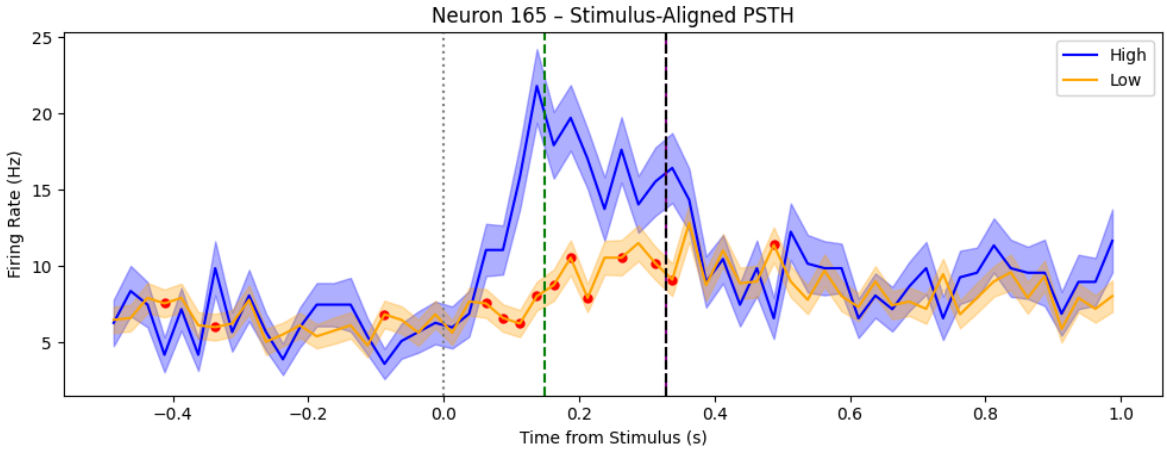

Neuron 147 – Stimulus-Aligned PSTH
High
Low
Firing Rate (Hz)
Time from Stimulus (s)

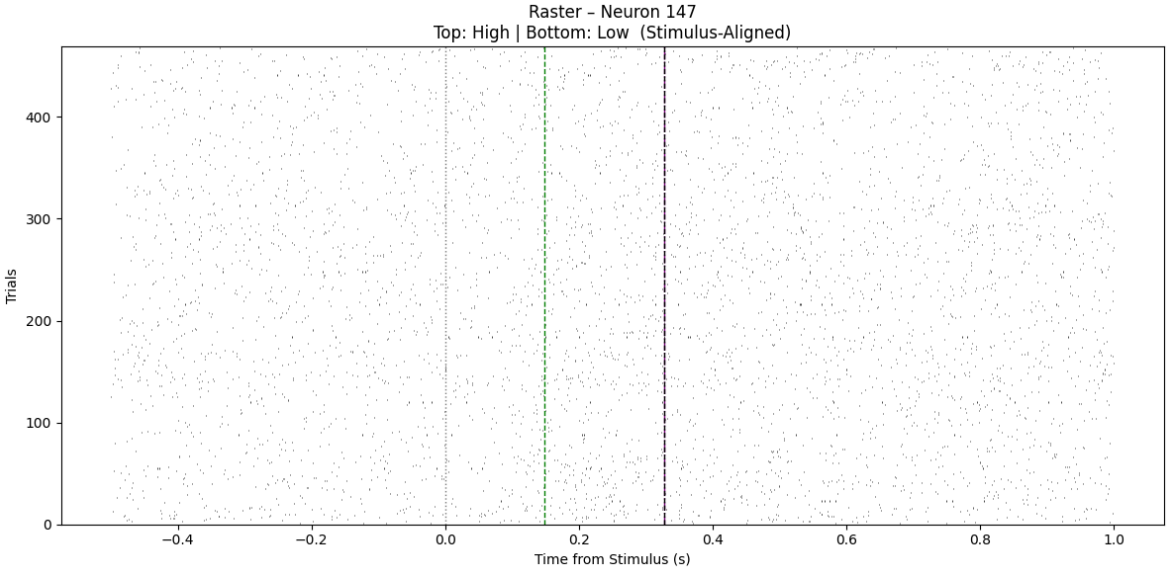

Neuron 155 – Stimulus-Aligned PSTH
High
Low
Firing Rate (Hz)
Time from Stimulus (s)

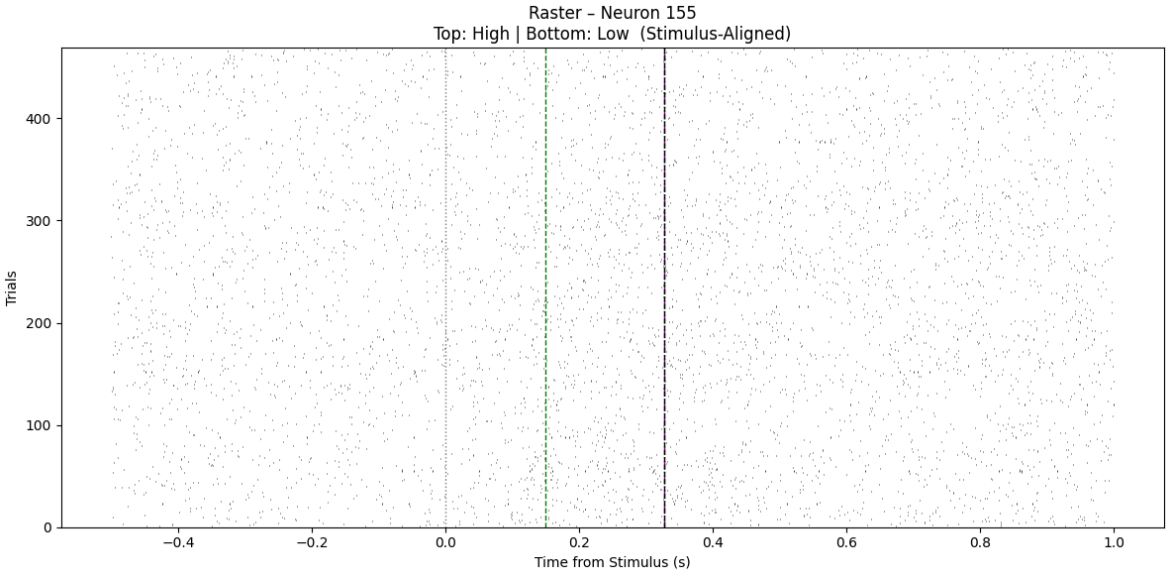

Neuron 160 – Stimulus-Aligned PSTH
High
Low
Firing Rate (Hz)
Time from Stimulus (s)

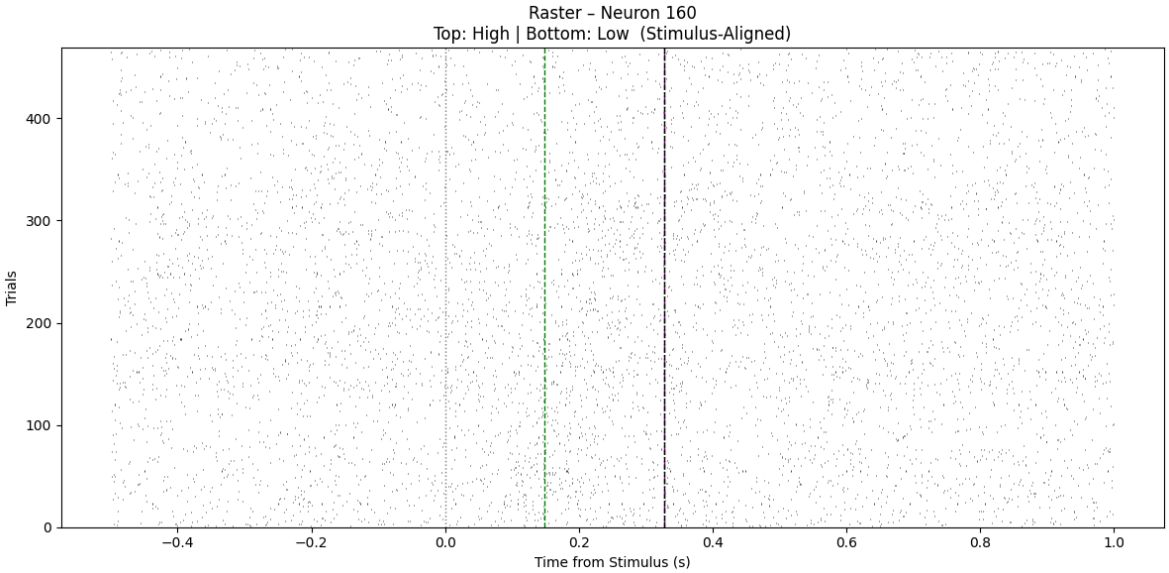

Neuron 162 – Stimulus-Aligned PSTH
High
Low
Firing Rate (Hz)
Time from Stimulus (s)

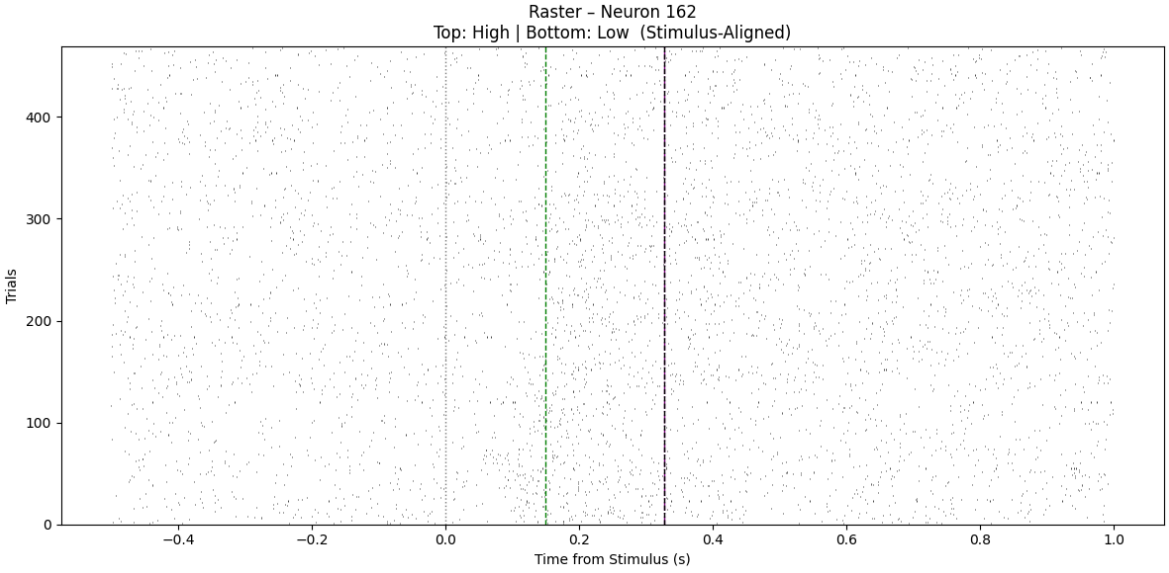

Neuron 164 – Stimulus-Aligned PSTH
High
Low
Firing Rate (Hz)
Time from Stimulus (s)

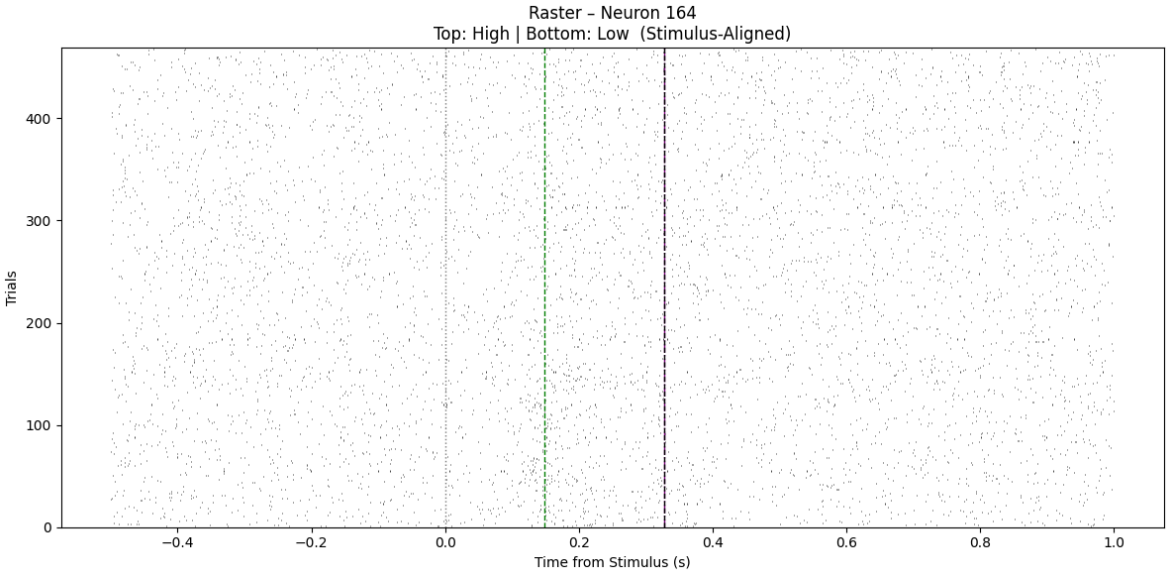

Neuron 165 – Stimulus-Aligned PSTH
High
Low
Firing Rate (Hz)
Time from Stimulus (s)

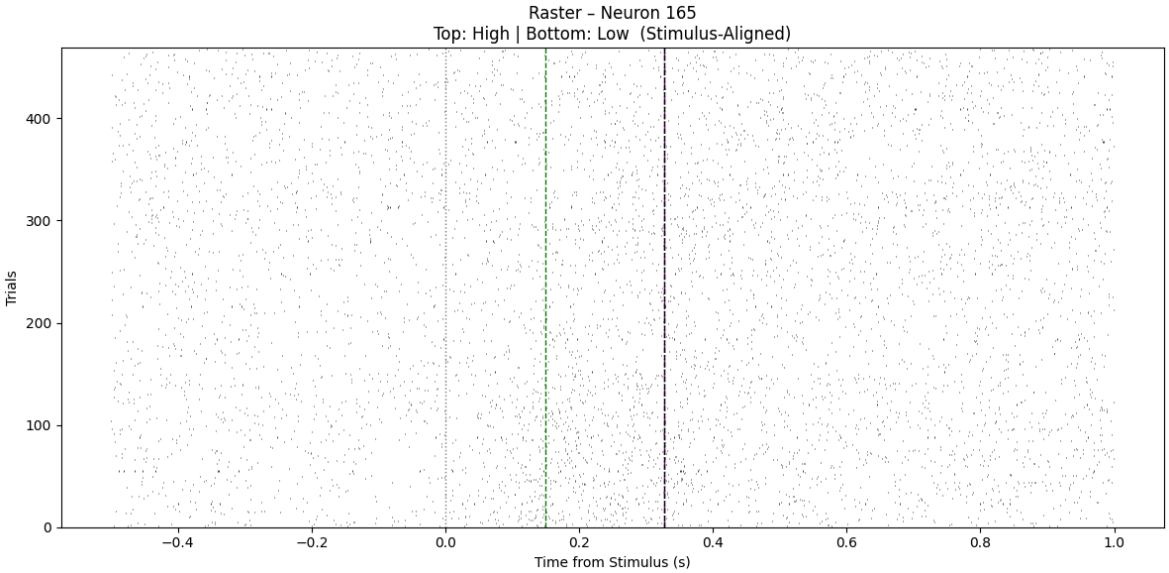

Raster – Neuron 147
Top: High | Bottom: Low (Stimulus-Aligned)
Trials
Time from Stimulus (s)


Raster – Neuron 155
Top: High | Bottom: Low (Stimulus-Aligned)
Trials
Time from Stimulus (s)


Raster – Neuron 160
Top: High | Bottom: Low (Stimulus-Aligned)
Trials
Time from Stimulus (s)


Raster – Neuron 162
Top: High | Bottom: Low (Stimulus-Aligned)
Trials
Time from Stimulus (s)






```
GROUP-LEVEL SUMMARY (Stimulus-Aligned)
 mean_high  mean_low  SEM_high  SEM_low  p_value  cohen_d
    0.2006     0.185    0.0021   0.0012      0.0 0.035164

PER-NEURON STATS
 neuron  mean_high  mean_low  p_value  cohen_d
    147     0.1643    0.1564 0.242937 0.019516
    155     0.1810    0.1745 0.131108 0.015590
    160     0.2292    0.2108 0.003971 0.039019
    162     0.1731    0.1680 0.556479 0.012214
    164     0.2178    0.2040 0.075490 0.029537
    165     0.2382    0.1965 0.000000 0.089092
```

In [35]:
```
# Reward aligned
# Parameters
selected_neurons = [147, 155, 160, 162, 164, 165]
pre_time = -0.5
post_time = 1.0

stim_onset = trials['stimOn_times'][correct_trials]
reward_onset = trials['feedback_times'][correct_trials]

# Prepare storage
firing_by_neuron = {}
stat_results = {}
```

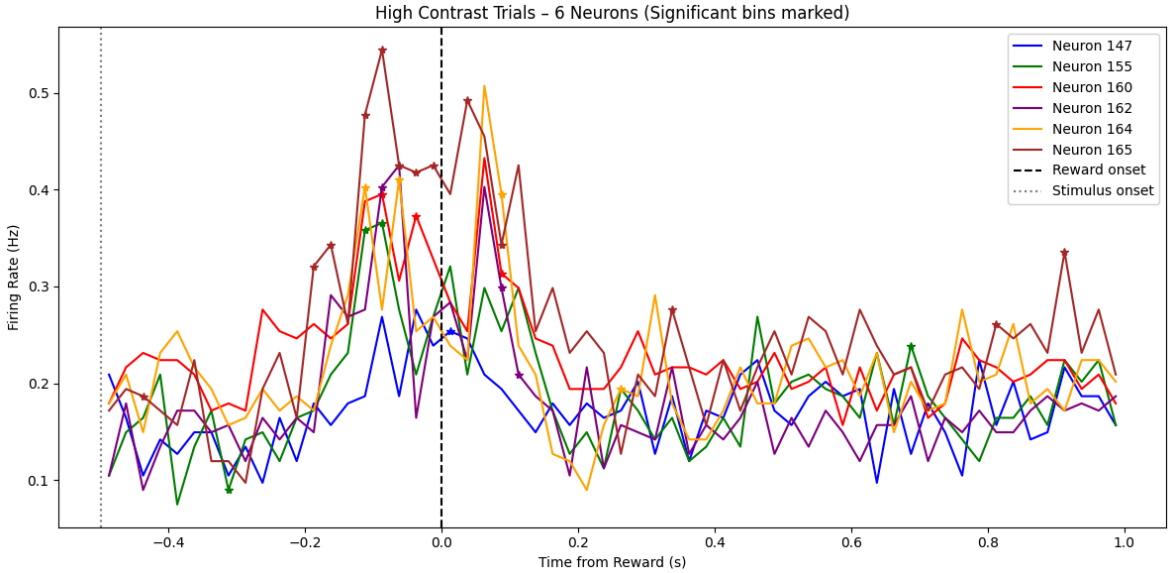
```
for neuron_id in selected_neurons:
    neuron_spikes = spikes['times'][spikes['clusters'] == neuron_id]
    high_matrix = []
    low_matrix = []
    p_values = []

    for t in reward_onset:
        aligned_spikes = neuron_spikes - t
        counts, _ = np.histogram(aligned_spikes, bins=T_BINS)
        idx = np.where(reward_onset == t)[0][0]
        if contrast_labels[idx] == 'high':
            high_matrix.append(counts)
        else:
            low_matrix.append(counts)

    high_matrix = np.array(high_matrix)
    low_matrix = np.array(low_matrix)
    firing_by_neuron[neuron_id] = {'high': high_matrix, 'low': low_matrix

    # Mann-Whitney U per bin
    bin_pvals = []
    for b in range(len(TIME_CENTERS)):
        if len(high_matrix) > 1 and len(low_matrix) > 1:
            stat, p = mannwhitneyu(high_matrix[:, b], low_matrix[:, b])
        else:
            p = 1.0
        bin_pvals.append(p)

    stat_results[neuron_id] = {
        'mean_high': high_matrix.mean(),
        'mean_low': low_matrix.mean(),
        'sem_high': sem(high_matrix),
        'sem_low': sem(low_matrix),
        'p_values': bin_pvals
    }

# Plot all neurons in one PSTH with sig markers
colors = ['blue', 'green', 'red', 'purple', 'orange', 'brown']
stim_onset_time = -0.5  # relative to reward
reward_onset_time = 0

#Plot High Contrast
plt.figure(figsize=(12, 6))
for i, neuron_id in enumerate(selected_neurons):
    fr_high = firing_by_neuron[neuron_id]['high']
    mean_high = fr_high.mean(axis=0)

    # Plot neuron
    plt.plot(TIME_CENTERS, mean_high, label=f'Neuron {neuron_id}', color=

    # Significance markers
    for t_idx, t in enumerate(TIME_CENTERS):
        fr_l = firing_by_neuron[neuron_id]['low'][:, t_idx]
        fr_h = firing_by_neuron[neuron_id]['high'][:, t_idx]
        if len(fr_l) > 5 and len(fr_h) > 5:
            _, p = mannwhitneyu(fr_l, fr_h)
            if p < 0.05:
                plt.plot(t, mean_high[t_idx], marker='*', color=colors[i]
```

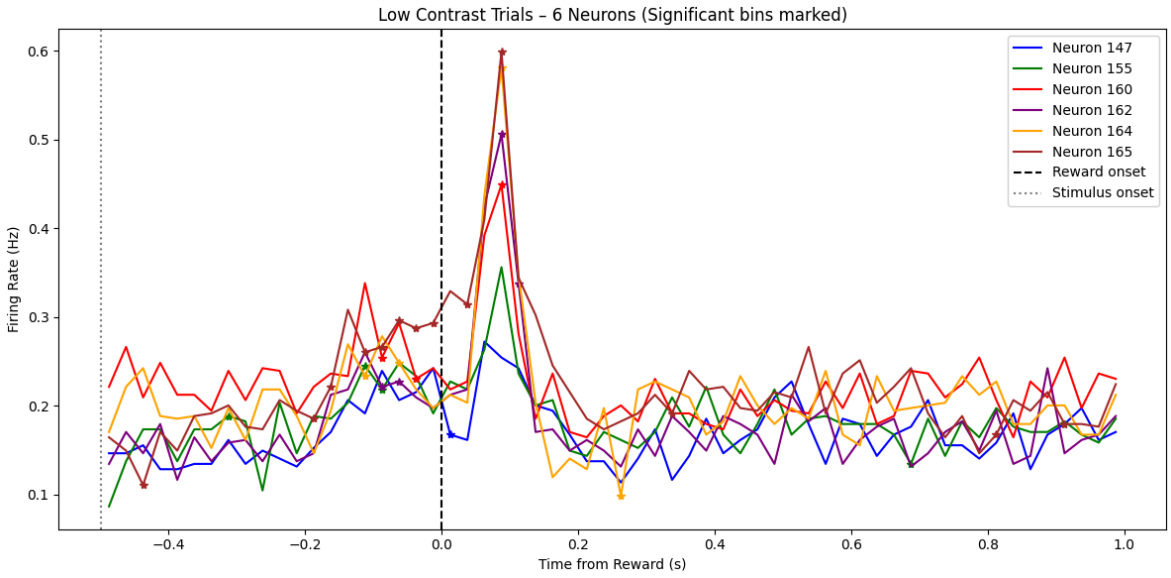
```
plt.axvline(reward_onset_time, linestyle='--', color='black', label='Rewa
plt.axvline(stim_onset_time, linestyle=':', color='gray', label='Stimulus
plt.title('High Contrast Trials — 6 Neurons (Significant bins marked)')
plt.xlabel('Time from Reward (s)')
plt.ylabel('Firing Rate (Hz)')
plt.legend()
plt.tight_layout()
plt.show()

# Plot Low Contrast
plt.figure(figsize=(12, 6))
for i, neuron_id in enumerate(selected_neurons):
    fr_low = firing_by_neuron[neuron_id]['low']
    mean_low = fr_low.mean(axis=0)

    # Plot neuron
    plt.plot(TIME_CENTERS, mean_low, label=f'Neuron {neuron_id}', color=c

    # Significance markers
    for t_idx, t in enumerate(TIME_CENTERS):
        fr_l = firing_by_neuron[neuron_id]['low'][:, t_idx]
        fr_h = firing_by_neuron[neuron_id]['high'][:, t_idx]
        if len(fr_l) > 5 and len(fr_h) > 5:
            _, p = mannwhitneyu(fr_l, fr_h)
            if p < 0.05:
                plt.plot(t, mean_low[t_idx], marker='*', color=colors[i],

plt.axvline(reward_onset_time, linestyle='--', color='black', label='Rewa
plt.axvline(stim_onset_time, linestyle=':', color='gray', label='Stimulus
plt.title('Low Contrast Trials — 6 Neurons (Significant bins marked)')
plt.xlabel('Time from Reward (s)')
plt.ylabel('Firing Rate (Hz)')
plt.legend()
plt.tight_layout()
plt.show()

# Individual neuron PSTHs with sig points

for i, neuron_id in enumerate(selected_neurons):
    fr_high = firing_by_neuron[neuron_id]['high']
    fr_low = firing_by_neuron[neuron_id]['low']
    mean_high = fr_high.mean(axis=0)
    mean_low = fr_low.mean(axis=0)
    sem_high = sem(fr_high, axis=0)
    sem_low = sem(fr_low, axis=0)

    plt.figure(figsize=(10, 4))
    plt.plot(TIME_CENTERS, mean_high, label='High', color='blue')
    plt.fill_between(TIME_CENTERS, mean_high - sem_high, mean_high + sem_
    plt.plot(TIME_CENTERS, mean_low, label='Low', color='orange')
    plt.fill_between(TIME_CENTERS, mean_low - sem_low, mean_low + sem_low

    sig_bins = np.where(np.array(stat_results[neuron_id]['p_values']) < 0
    plt.scatter(TIME_CENTERS[sig_bins], mean_low[sig_bins], color='red',

    plt.axvline(0, linestyle='--', color='k')
    plt.axvline(-reward_onset[0] + stim_onset[0], linestyle=':', color='g
    plt.title(f'Neuron {neuron_id} Firing Rate')
    plt.xlabel('Time (s)')
    plt.ylabel('Firing Rate (Hz)')
```

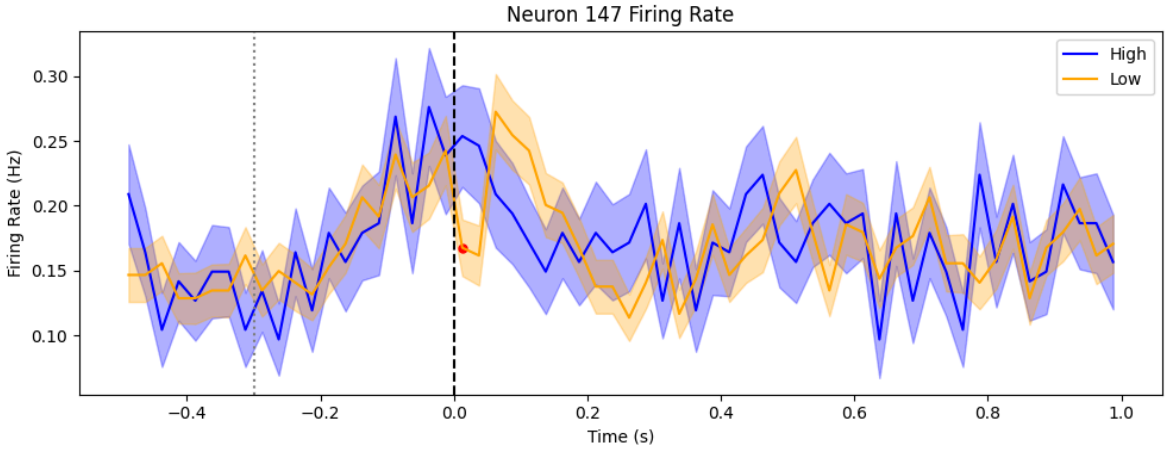
```
    plt.legend()
    plt.tight_layout()
    plt.show()


#  Raster plots (high vs low)
# Parameters
time_window = (-0.5, 1.0)
correct_trials_mask = trials['feedbackType'] == 1
reward_times = trials['feedback_times'][correct_trials_mask]
contrast = trials['contrastLeft'].copy()
contrast[np.isnan(contrast)] = trials['contrastRight'][np.isnan(contrast)
contrast_labels = np.where(contrast == 1.0, 'high', 'low')
contrast_labels = contrast_labels[correct_trials_mask]
valid_trials = np.where(correct_trials_mask)[0]

raster_by_neuron = {}

for neuron_id in selected_neurons:
    spikes_neuron = spikes['times'][spikes['clusters'] == neuron_id]

    raster_by_neuron[neuron_id] = {'high': [], 'low': []}

    for i, trial_idx in enumerate(valid_trials):
        reward_time = trials['feedback_times'][trial_idx]
        aligned_spikes = spikes_neuron - reward_time
        spikes_in_window = aligned_spikes[
            (aligned_spikes >= time_window[0]) & (aligned_spikes <= time_
        ]
        trial_label = contrast_labels[i]
        raster_by_neuron[neuron_id][trial_label].append(spikes_in_window)


stim_onset_time = -0.5
reward_onset_time = 0

for neuron_id in selected_neurons:
    high_trials = raster_by_neuron[neuron_id]['high']
    low_trials = raster_by_neuron[neuron_id]['low']
    fr_high = firing_by_neuron[neuron_id]['high']
    fr_low = firing_by_neuron[neuron_id]['low']

    plt.figure(figsize=(12, 6))

    # Plot raster for high contrast (top)
    for i, trial in enumerate(high_trials):
        plt.vlines(trial, i + 0.5, i + 1.5, color='black', linewidth=0.5)

    high_end = len(high_trials)

    # Plot raster for low contrast (bottom)
    for i, trial in enumerate(low_trials):
        plt.vlines(trial, high_end + i + 0.5, high_end + i + 1.5, color='

    total_trials = high_end + len(low_trials)

    # Mark stimulus and reward times
    plt.axvline(stim_onset_time, linestyle=':', color='black', linewidth=
    plt.axvline(reward_onset_time, linestyle='--', color='black', linewid
```

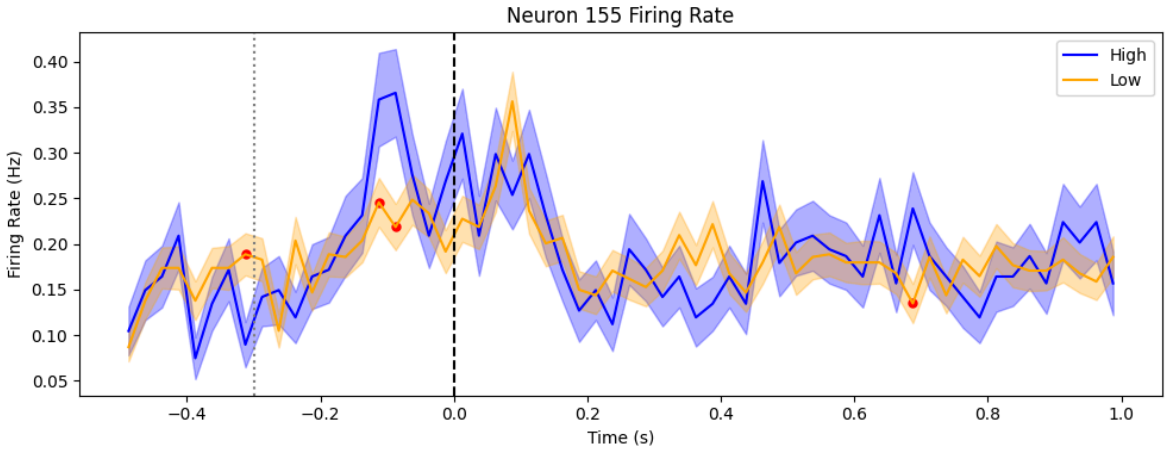
```
    # Run per-bin statistical test and mark significant bins
    for t_idx, t in enumerate(TIME_CENTERS):
        h = fr_high[:, t_idx]
        l = fr_low[:, t_idx]
        if len(h) > 5 and len(l) > 5:
            _, p = mannwhitneyu(h, l)
            if p < 0.05:
                plt.axvline(t, color='red', linestyle='-', linewidth=1)

    plt.ylim(0, total_trials + 1)
    plt.title(f'Raster Plot — Neuron {neuron_id}\nTop: High Contrast | Bo
    plt.xlabel('Time from Reward (s)')
    plt.ylabel('Trials')
    plt.tight_layout()
    plt.show()


# Prepare group-level storage
all_high = []
all_low = []

# For per-neuron results
results = []

for neuron_id in selected_neurons:
    fr_high = np.array(firing_by_neuron[neuron_id]['high'])
    fr_low = np.array(firing_by_neuron[neuron_id]['low'])

    # Store for group stats
    all_high.extend(fr_high)
    all_low.extend(fr_low)

   #stats
fr_high = np.array(firing_by_neuron[neuron_id]['high'])
fr_low = np.array(firing_by_neuron[neuron_id]['low'])

all_high.extend(fr_high)
all_low.extend(fr_low)

stat, p_val = mannwhitneyu(fr_high.flatten(), fr_low.flatten(), alternati
mean_high = np.mean(fr_high)
mean_low = np.mean(fr_low)
pooled_std = np.sqrt((np.var(fr_high, ddof=1) + np.var(fr_low, ddof=1)) /
cohen_d = (mean_high - mean_low) / pooled_std if pooled_std > 0 else 0

results.append({
    'neuron': neuron_id,
    'mean_high': round(mean_high, 4),
    'mean_low': round(mean_low, 4),
    'p_value': round(float(p_val), 6),
    'cohen_d': round(cohen_d, 6)
})


# Group-level stats
all_high = np.array(all_high)
all_low = np.array(all_low)
group_mean_high = np.mean(all_high)
group_mean_low = np.mean(all_low)
group_stat, group_p = mannwhitneyu(np.array(all_high).flatten(), np.array
```

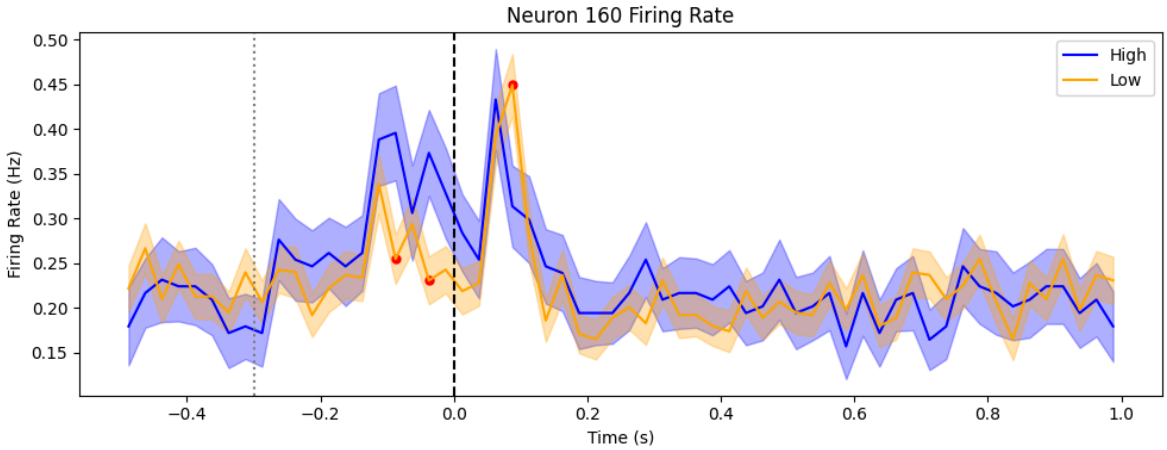
```
group_pooled_std = np.sqrt((np.var(all_high, ddof=1) + np.var(all_low, dd
group_cohen_d = (group_mean_high - group_mean_low) / group_pooled_std if

group_result = {
    'mean_high': round(group_mean_high, 4),
    'mean_low': round(group_mean_low, 4),
    'SEM_high': round(np.std(all_high, ddof=1) / np.sqrt(len(all_high)),
    'SEM_low': round(np.std(all_low, ddof=1) / np.sqrt(len(all_low)), 4),
    'p_value': round(float(group_p), 6),
    'cohen_d': round(group_cohen_d, 6)
}

# Convert to DataFrames
per_neuron_df = pd.DataFrame(results)
group_summary_df = pd.DataFrame([group_result])
# Display the tables
print("=== GROUP-LEVEL SUMMARY (Cluster 2)")
print(group_summary_df.to_string(index=False))
print("\n Per-neuron Stats (Cluster 2)")
print(per_neuron_df.to_string(index=False))
```

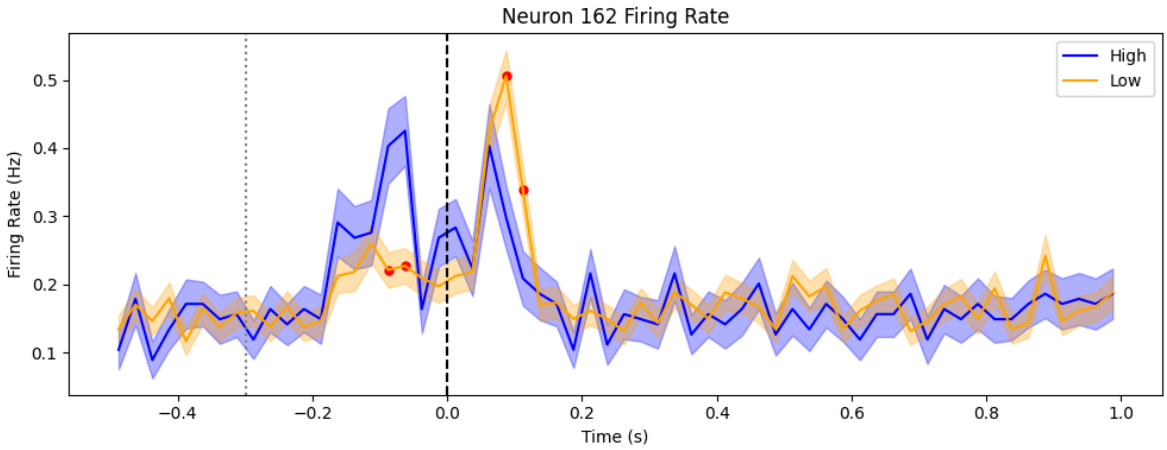


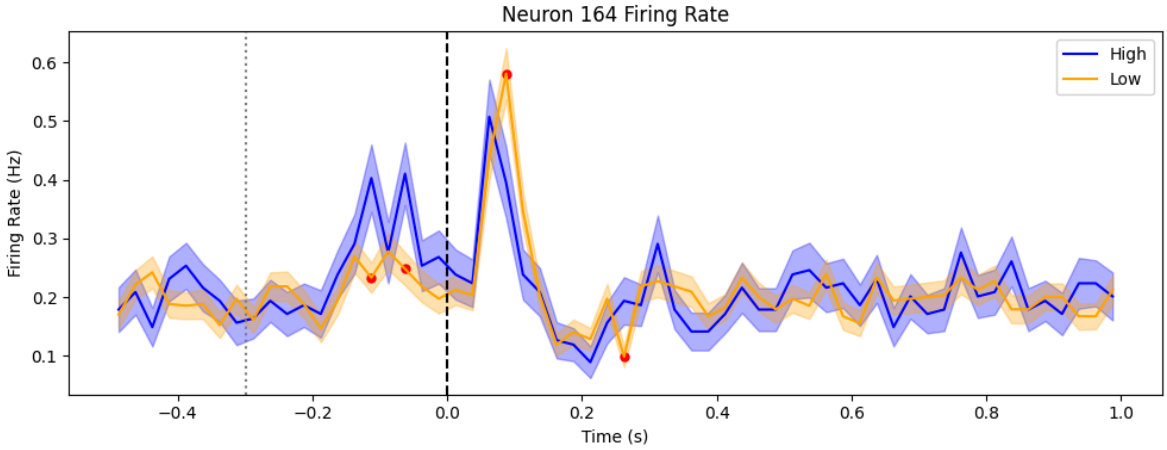


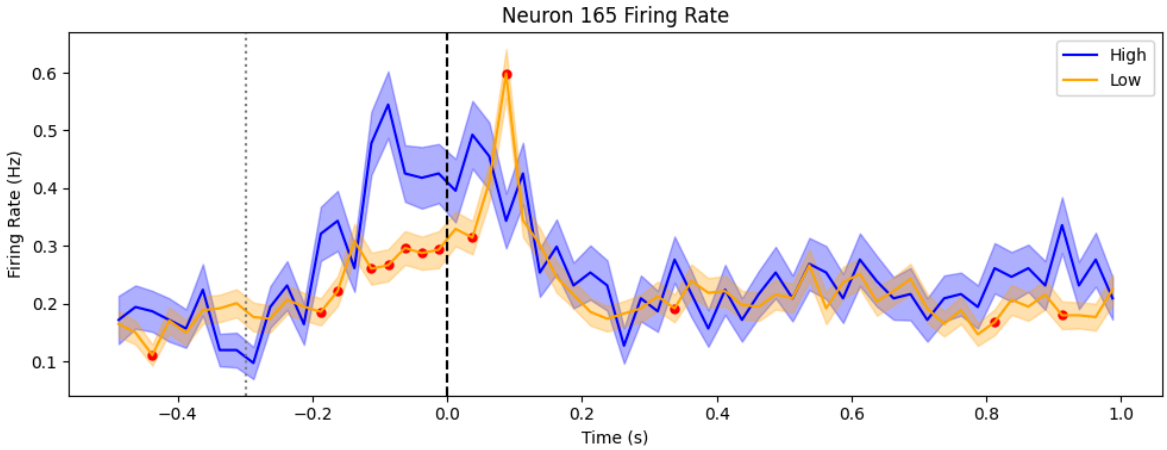

Neuron 147 Firing Rate
High
Low
Firing Rate (Hz)
Time (s)

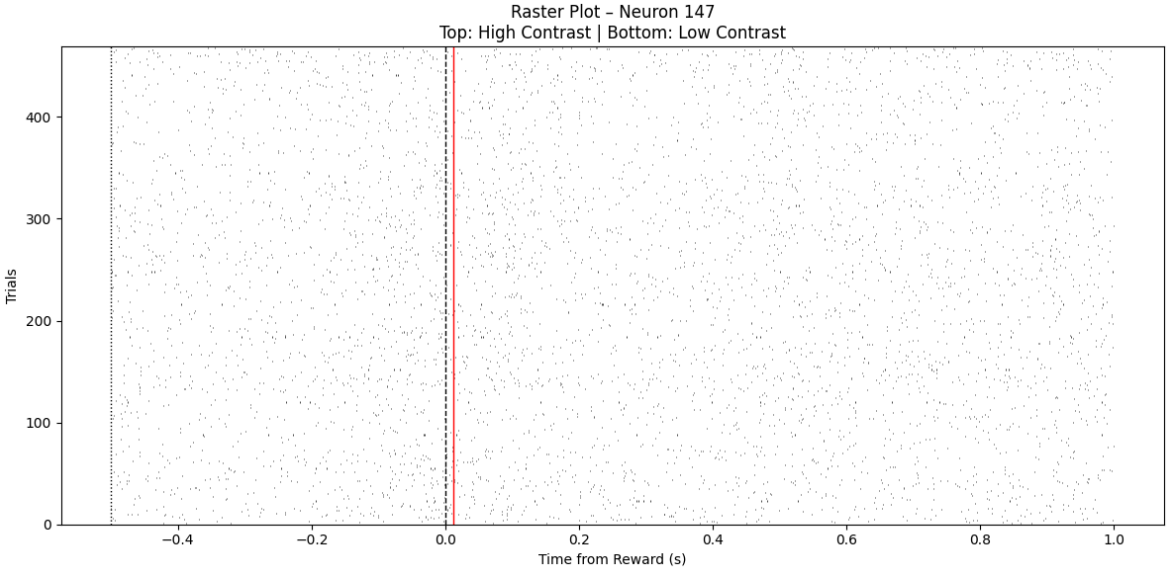

Neuron 155 Firing Rate
High
Low
Firing Rate (Hz)
Time (s)

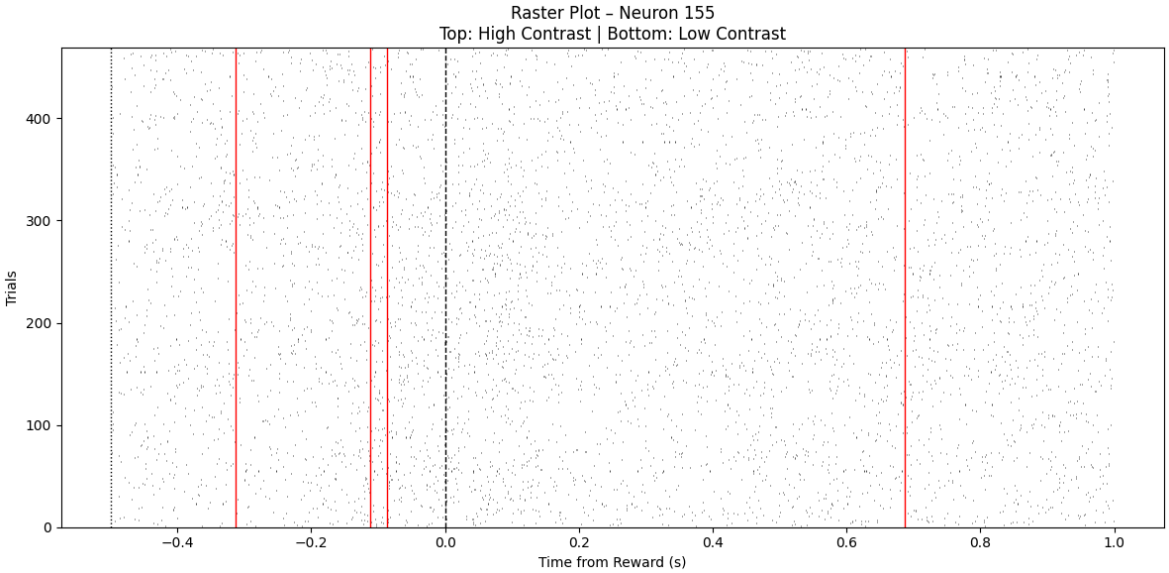

Neuron 160 Firing Rate
High
Low
Firing Rate (Hz)
Time (s)

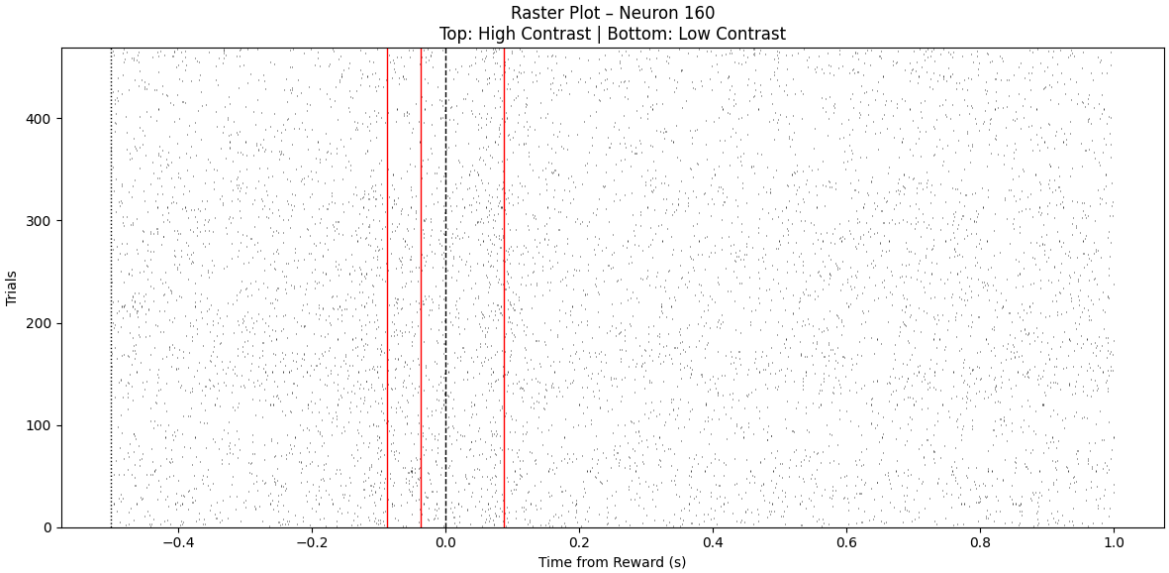

Neuron 162 Firing Rate
High
Low
Firing Rate (Hz)
Time (s)

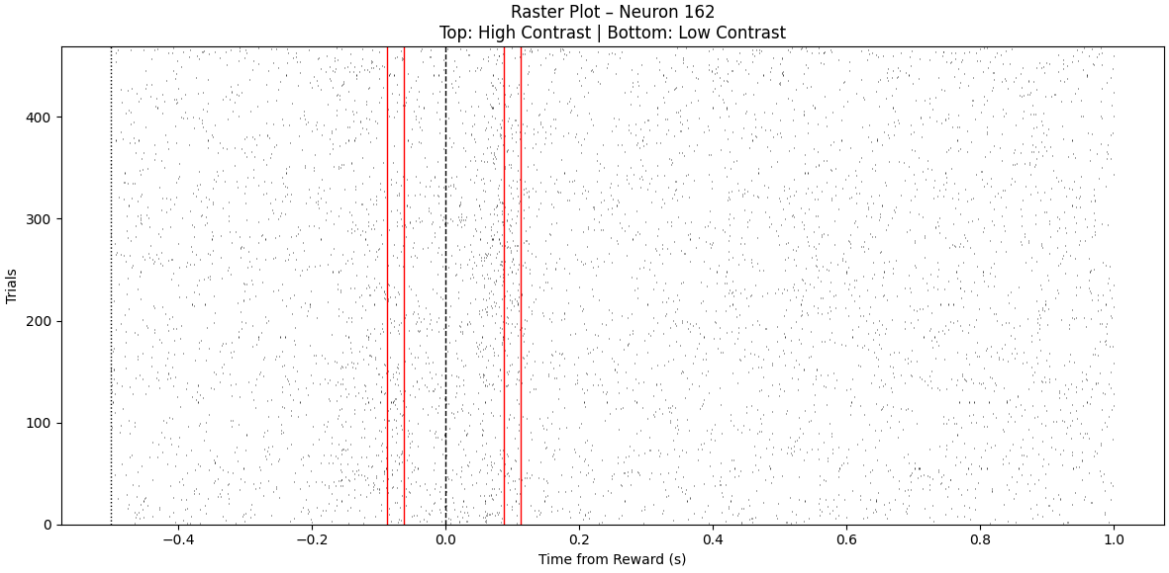

Neuron 164 Firing Rate
High
Low
Firing Rate (Hz)
Time (s)

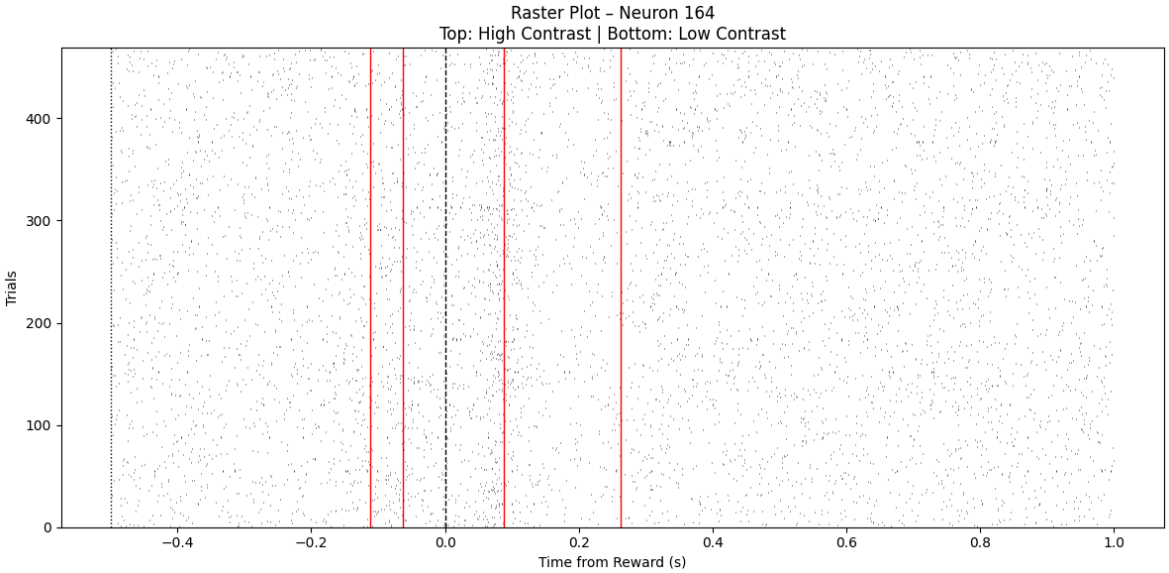

Neuron 165 Firing Rate
High
Low
Firing Rate (Hz)
Time (s)

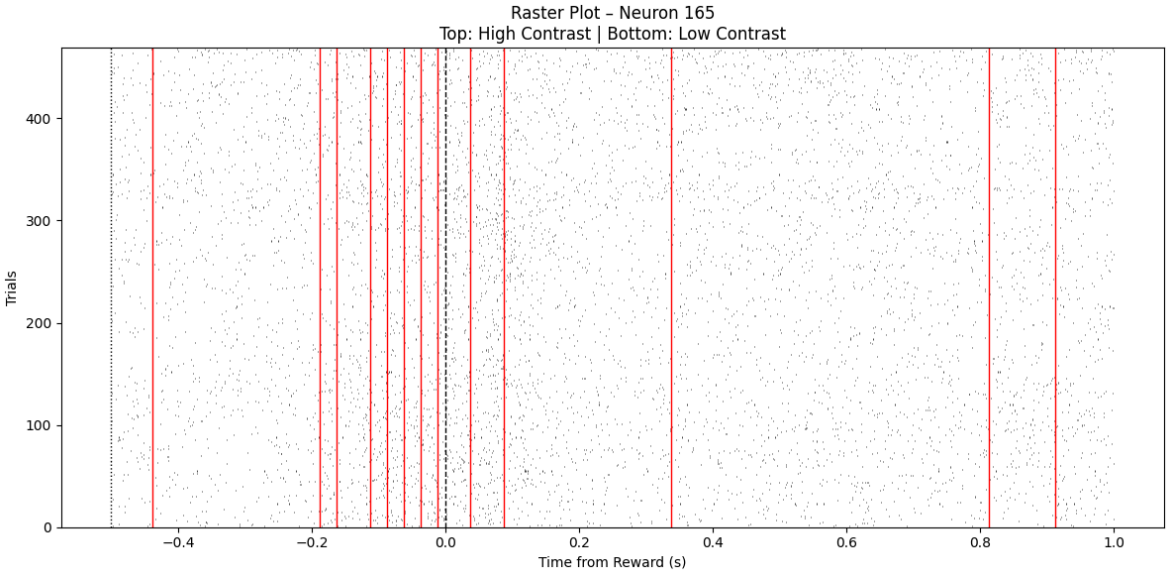

Raster Plot – Neuron 147
Top: High Contrast | Bottom: Low Contrast
Trials
Time from Reward (s)


Raster Plot – Neuron 155
Top: High Contrast | Bottom: Low Contrast
Trials
Time from Reward (s)


Raster Plot – Neuron 160
Top: High Contrast | Bottom: Low Contrast
Trials
Time from Reward (s)


Raster Plot – Neuron 162
Top: High Contrast | Bottom: Low Contrast
Trials
Time from Reward (s)






```
=== GROUP-LEVEL SUMMARY (Cluster 2)
 mean_high  mean_low  SEM_high  SEM_low  p_value  cohen_d
    0.2153    0.2032    0.0153   0.0094 0.000001 0.026337

 Per-neuron Stats (Cluster 2)
 neuron  mean_high  mean_low  p_value  cohen_d
    165     0.2565    0.2241 0.000001 0.066085
```

Conclusion:

In this project, I investigated how dopaminergic neurons in the Ventral Tegmental Area (VTA) encode reward prediction errors (RPEs) under varying sensory uncertainty. My main focus was to compare VTA responses between high- and low-contrast correct trials to test whether reward-related neural activity scales with the level of perceptual uncertainty. I hypothesized that correct low-contrast trials would evoke stronger dopaminergic responses due to higher reward unexpectancy, while correct high-contrast trials would show weaker responses reflecting higher reward expectation. To do this, I aligned spike data to task events, computed peristimulus time histograms (PSTHs), and performed both population-level and single-neuron statistical analyses. While my original plan also included testing a second hypothesis

related to reward omission in incorrect high-contrast trials, time constraints limited the scope of the analysis to only rewarded trials.