SYNAPSE: A Multi-Modal Framework for Interpretable Neural Decoding Using Vision-Language Foundation Models

Edward Ju [*1] **Adarsh Kumarappan** [*1] **Shrujana Kunnam** [*1] **Raaghav Malik** [*1] **Dhruv Sheth** [*1]

Abstract

Understanding how the human brain represents rich, dynamic visual experiences requires feature spaces that are both interpretable and decodable from neural activity. Leveraging large language models to generate cross-modal embeddings, we present SYNAPSE (SYstematic Neural Analysis through Prompt-based Semantic Extraction), a GPT-4o-based framework for automatic extraction and categorization of semantically meaningful features from visual stimuli. SYNAPSE generates interpretable embeddings spanning emotions, actions, objects, and cinematographic features - everything from heart-racing tension and joyful laughter to dramatic close-ups and camera movements - all without manual labeling. We evaluate our approach by automatically extracting and then decoding features from single-neuron activity during an 8-minute movie clip, using four machine learning models: logistic regression, a multilayer perceptron, a long-short term memory network, and a bidirectional long-short term memory network. From comparison with manual labels, we find that our automatic labels accurately identify salient features with a false positive error rate under 1.5%, and our results show that our framework is reliably able to determine what features are decodable across models, demonstrating the viability of our proposed framework in automatically discovering and generating interpretable representations of dynamic information aligned with neural activity.

[*]Equal contribution [1]Department of Computing + Mathematical Sciences, California Institute of Technology, Pasadena, CA, United States. Correspondence to: Adarsh Kumarappan <akumarap@caltech.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

1. Introduction

Building a mechanistic understanding of how the human brain processes and encodes complex stimuli remains a fundamental challenge in neuroscience. Advances in neural recording technologies have enabled the acquisition of human single-neuron recordings to rich, dynamic stimuli, such as movies (Engel et al., 2005). These naturalistic inputs, which contain intricate, time-evolving visual, auditory, and semantic content, present a unique opportunity for studying brain function in real-world contexts but prove difficult to characterize and quantify in relation to neural activity.

Current approaches to neural decoding of complex stimuli rely on training models on manually defined feature sets, which often do not capture the full range of relevant features that drive neural responses. These methods often require hand-annotated labels or focus on specific stimulus attributes (e.g., object recognition or face detection) rather than providing a comprehensive framework for analyzing multiple feature dimensions simultaneously (Glaser et al., 2020). Low-dimensional feature spaces are limited in their generalizability and fail to capture the depth of information present in naturalistic stimuli, hindering our ability to study how different brain regions process and integrate various aspects of real-world experiences.

In this work, we present a framework for analyzing neural responses to visual stimuli by leveraging large language models (LLMs) to automatically extract and categorize movie features. Specifically, we leverage the vision-language capabilities of GPT-4o (OpenAI et al., 2024) to generate detailed descriptions of movie frames which we convert into interpretable embeddings, enabling human readability and consequent neural analysis.

We demonstrate its efficacy by decoding a diverse array of features, including emotions, actions, shot types, and scene composition, from single-neuron activity of patients watching a film clip. Using the extracted features, we train and evaluate decoding performance of four machine learning (ML) models: logistic regression, a multilayer perceptron (MLP), a long-short term memory (LSTM) network, and a bidirectional LSTM with attention (BiLSTM with attention) network. Our tool allows for streamlined reconfiguration through simple prompt modification to extract custom features sets and efficiently train diverse models in neuroscience and broader contexts.

2. Related Work

2.1. Neural Decoding with Naturalistic Stimuli

Early work investigating how the brain represents static visual stimuli has identified cells in the human medial temporal lobe (MTL) that respond selectively to specific objects, faces, or people (Kreiman et al., 2000), including single neurons that encode the specific identity of individual stimuli (Quiroga et al., 2005). More recent studies have turned to naturalistic stimuli, such as movies, to better capture how the brain processes information under dynamic conditions. Intracranial recordings during movie-viewing have revealed that population-level neural activity encodes high-level features such as character identity, scene location, and visual transitions, which can be decoded with machine learning models (Gerken et al., 2024). Most of these studies rely on manual annotations or narrowly trained machine learning models for object and character recognition, which is both resource-intensive and limits scalability across varied contexts. To address this limitation, our work leverages LLMs to generate generalizable and interpretable embeddings directly from visual content, eliminating the need for hand-annotated labels or task-specific training. This enables larger and more systematic investigations into how individual neurons and neural populations represent features of naturalistic stimuli.

2.2. Deep Learning Methods in Neural Decoding

Deep learning models, particularly recurrent neural networks (RNNs), long short-term memory networks (LSTMs), and transformers, have demonstrated strong performance in neural decoding tasks requiring temporal sensitivity and robust feature representation (Zhang et al., 2023). LSTMs outperform traditional machine learning models in neural decoding tasks involving temporal dependencies, such as predicting motor trajectories (Liu et al.), highlighting their potential for decoding dynamic stimuli such as continuous movie content. Past research comparing machine learning models for decoding movie features has shown that simpler models, such as logistic regression, often perform comparably to deep learning models while being less computationally-expensive (Gerken et al., 2024). This motivates the application of our framework to various model architectures, highlighting its flexibility and applicability across decoding contexts.

2.3. Interpretable Embeddings from Large Language Models

Recent advancements in LLMs offer a promising alternative for creating text embeddings, capturing complex relationships to enable higher quality semantic representations (Wang et al., 2024) . However, these tools lack interpretability, making them challenging for neuroscience where understanding the underlying mechanisms of the brain requires comprehensible feature spaces. The innovation of interpretable question-answer-based LLM embeddings in language neuroscience (Benara et al., 2024) provides a framework that we extend to visual stimuli, offering a more generalizable approach for interpreting neural activity. By leveraging the multi-modal capabilities of GPT-4o, our approach enables the generation of comprehensive, structured, and semantically meaningful representations of visual information for human-interpretable neural analysis.

3. Framework

Our framework is designed to automatically sift through a large pool of candidate features—drawn from rich, language-model–generated embeddings—and identify which of those features are reliably present in single-neuron activity. Rather than hand-selecting a handful of features for decoding, our pipeline efficiently evaluates every feature in parallel, flagging only those that achieve above-chance prediction performance, enabling scalable supervised learning from unlabeled data. Our method consists of the following steps:

1. **Automatic Description Generation:** We use GPT-4o via the GPT-4o API for frame-by-frame feature extraction, enabling structured semantic analysis of visual content. Frames are sampled, saved as JPEGs, and processed with prompts configured for concise, focused API responses. Two tailored prompts are used for each frame: one to capture surface-level elements such as objects, actions, and emotions, and another for deeper narrative and cinematic analysis. These prompts evolved iteratively to balance structured formats with rich details, ensuring consistent, high-quality outputs. The resulting natural language descriptions capture visible scene elements, dynamic movements, and multi-dimensional emotional indicators, such as facial expressions, interactions, and mood.

2. **Label Extraction:** From the generated descriptions, binary word labels are extracted, capturing relevant features (e.g., man, tree, car) present in the images. The labels are systematically organized based on recurrence patterns (e.g., appearing in at least 10% of the frames).

3. **Dataset Construction:** The binary word labels are structurally post-processed through synonym grouping

to build a dataset without requiring manual annotation. The dataset is stored in an efficient HDF5 format with features stored hierarchically. Validation and error handling occur at each step, ensuring data integrity.

4. **Model Training:** The dataset can then be used in conjunction with biological data, such as neural activity, to predict the presence or absence of each concept. This enables the decoding of semantic information from biological activity using automated and interpretable feature embeddings.

4. Evaluation

4.1. Neural Dataset

We use a public dataset from (Keles et al., 2024) consisting of 16 epilepsy patients undergoing clinical monitoring at Cedars-Sinai Medical Center. Over 29 sessions, patients watched an 8-minute black-and-white episode of *Alfred Hitchcock Presents: Bang! You're Dead* while single-unit, LFP, and iEEG data were recorded from hybrid depth electrodes in medial temporal and frontal areas. This stimulus is widely used due to its high inter-subject correlation across cortical regions (Hasson et al., 2009). Neural signals were acquired at high sampling rates and preprocessed using spike-sorting and electrode localization.

4.2. Data Processing

We incorporate temporal context and guard against data leakage by combining two complementary strategies. First, following the standard "temporal window" approach, each prediction at frame i includes its preceding (window – 1) frames as input; we evaluate window sizes of 1, 5, 10, and 20 for all three models (MLP, logistic, and LSTM). Second, because adjacent movie frames are often nearly identical (e.g., a static gun shot spanning many frames), randomly splitting individual frames can leak information across train/val/test sets and inflate performance. To address this, we break the video into contiguous blocks of size 1, 10, 50, 100, 200, 500, 1000, or 2000 frames, then randomly assign entire blocks to 70% train, 15% validation, and 15% test. By training a logistic regression on human binary labels over 1000 random splits, we observe that validation and test accuracy stabilize once blocks reach 1000 frames, indicating this block size best balances leakage prevention with sufficient label diversity.

Because the prevalence of positive labels varies widely across word-level features, we also create balanced versions of each split via oversampling. After dividing into temporal blocks and assigning them to train/val/test, we identify the minority class in each set and replicate its samples until its count matches the majority. When using temporal windows, we oversample entire sequences (frame i through i + T – 1) whose terminal label is minority until sequence-level balance is achieved. This process yields both original and balanced datasets for every split, ensuring models are trained and evaluated under comparable class distributions.

4.3. Model Architecture

We train four standard models—logistic regression, MLP, LSTM, and BiLSTM with attention—to determine whether each decodable feature's neural representation is expressed via linear, nonlinear, or temporal patterns. This allows our framework to both automatically identify decodable features and the computational form of their neural representations. While simpler models like logistic regression help establish baseline decodability and linear separability of features, more sophisticated architectures like LSTMs and BiLSTMs with attention mechanisms are employed to capture potential temporal integration and complex patterns in neural firing rates that might align with movie features. Implementation and architecture details are described in 9.4.

4.4. Assessment Metrics

We evaluate model performance on two separate test sets to address label imbalance. On a balanced test set, we report accuracy (overall fraction correct) and balanced accuracy (average recall across classes). On the naturally unbalanced set, we use Cohen's κ, F1 score, AUROC, and PR AUC to capture chance-corrected agreement, precision–recall tradeoffs, and discrimination ability.

To ensure stability across temporal splits, we run a 3-seed cross-validation: for each of three random temporal-block partitions, we train and evaluate the model and then aggregate each metric's mean and standard deviation.

For statistical validation, we add two circular-shift label shuffles per seed (random shifts of 2,000–10,000 frames selected once), yielding six control runs that preserve neural autocorrelation but destroy feature alignment. We then compare the real (n = 3) versus control (n = 6) metric distributions using a one-sided Welch's t-test, computing control variance by pooling both between-shuffle and within-shuffle variability, to test whether decoding performance exceeds chance.

5. Experiments and Results

5.1. Feature Extraction Accuracy

To evaluate the quality of our framework's feature extraction, we compared its results against manual annotations for several of the most frequently occurring labels in our dataset, including *close-up*, *child*, *gun*, and *hat*. Overall, the model achieves strong performance, with particularly high agreement on visually distinctive features such as *hat*, which is predicted with 92.1% accuracy, where most of the errors are false negatives where *hat* is present in the background of the frame but not significant (1).

Across all categories, false positives are rare - typically under 1.5% and as low as 0.07% for *child* - suggesting that GPT-4o's representations effectively favor perceptually grounded features and avoid hallucinating extraneous content. Slight disagreements arise from false negatives, in which a feature is present in the background of a frame but not labeled as important by the model. Given the context of neural decoding, this pattern proves favorable: our framework is designed to identify the most salient aspects of each frame, mirroring the brain's tendency to focus on behaviorally relevant cues.

5.2. Single-Neuron Decoding Performance

To assess how neural signals encode common perceptual features, we perform binary classification on a subset of the most cinematically meaningful and commonly extracted labels of our framework. These include objects and visual elements (*gun*, *close-up*), actions and body states (*standing*, *holding*), and emotional or abstract features (*innocent*, *happy*).

We train four classifiers - logistic regression, MLP, LSTM, and BiLSTM with attention - to decode the presence or absence of these features from neural activity. We both report overall decoding accuracy across selected features, providing a broad comparison of model performance. While not all features are equally decodable, some show stronger signal, particularly those that are more visually prominent and may be more robustly encoded in neural representations. We then compare each of these to the controls, showing that close-up achieves decoding accuracy significantly above chance across models, with improved performance for nonlinear and temporal decoding models (Figure 2). This feature's high decoding performance highlights the potential of our framework to capture meaningful neural correlates of high-level visual concepts. These results further underscore the flexibility of our framework to different model architectures, demonstrating its robustness across linear, nonlinear, and temporal decoding pipelines. Full results, including all p-values and assessment of other metrics, are described in Appendix 9.4.

6. Conclusion

In this work, we introduce SYNAPSE, a multi-modal framework that leverages the vision-language capabilities of GPT-4o to automatically extract interpretable, semantically meaningful features from movie stimuli. By removing the need for manual annotation, SYNAPSE reduces labor-intensive labeling and potential annotator bias, while enabling scalable and generalizable neural decoding. Our results demonstrate that certain visually salient features, such as *close-up*, can be robustly decoded from single-neuron activity, especially using nonlinear and temporal decoding models. While our evaluation focuses on a specific subset of features and a single dataset, SYNAPSE is inherently extensible to a broad range of visual concepts, neural modalities, and model architectures. This flexibility enables researchers to meaningfully investigate the neural basis of naturalistic perception, contributing to our growing understanding of how the human brain processes and encodes complex stimuli. Future work will aim to validate this framework on larger neural datasets with more complex and diverse visual stimuli, which may improve decoding performance of extracted feature sets and further enhance our understanding of human neural representation.

7. Impact Statement

This paper introduces a framework for automatically extracting language-based features. By leveraging large vision-language models to replace manual annotation, SYNAPSE reduces human bias and enhances scalability in neural decoding research. However, we acknowledge that such models may themselves encode bias from their training data, and care should be taken when interpreting features in sensitive contexts. Our framework is currently applied post hoc to publicly available, anonymized human neural data, but future applications should be developed with careful consideration to privacy, informed consent, and ethical handling of neural data. We hope our framework enables more interpretable, ethical approaches to studying human brain activity.

8. Acknowledgments

We would like to express our gratitude to Kevin J. M. Le, Geeling Chau, and Dr. Katherine L. Bouman for their invaluable mentorship, support, and guidance.

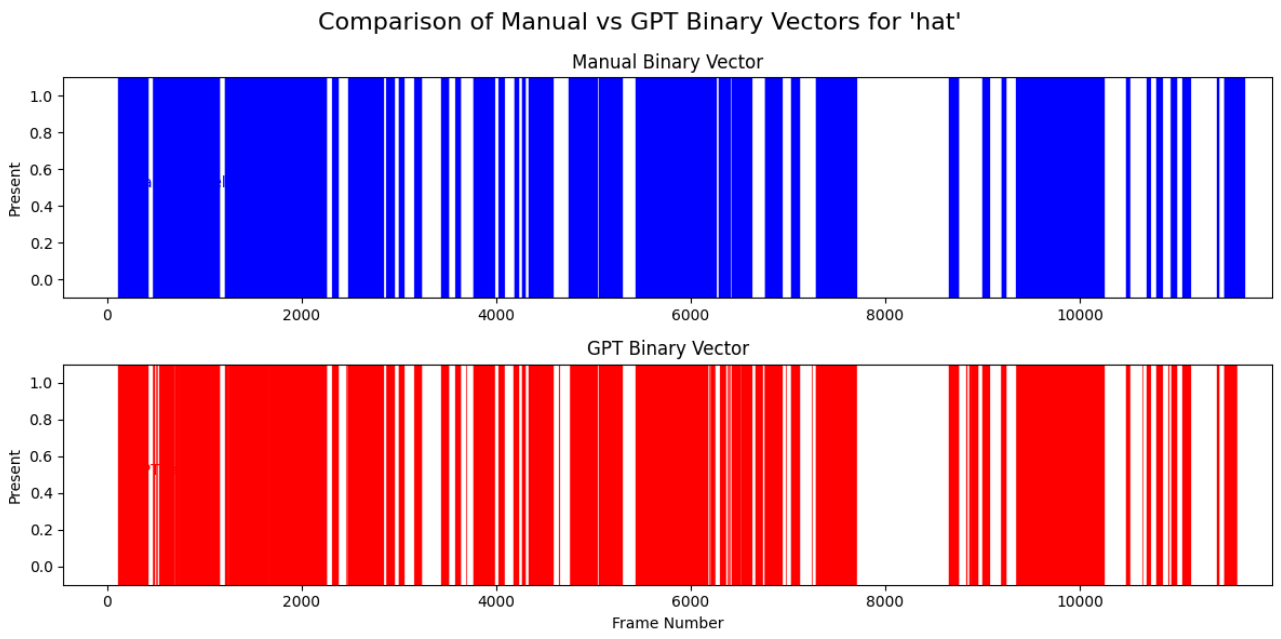

Figure 1: Raster plot comparison of GPT-encoded hat labels with manual labels for movie frames.

References

Vinamra Benara, Chandan Singh, John X. Morris, Richard Antonello, Ion Stoica, Alexander G. Huth, and Jianfeng Gao. Crafting interpretable embeddings by asking llms questions, 2024. URL `https://arxiv.org/abs/2405.16714`.

Andreas K. Engel, Christian K. E. Moll, Itzhak Fried, and George A. Ojemann. Invasive recordings from the human brain: clinical insights and beyond. *Nature Reviews Neuroscience*, 6 (1):35–47, Jan 2005. ISSN 1471-0048. doi: 10.1038/nrn1585. URL `https://doi.org/10.1038/nrn1585`.

Franziska Gerken, Alana Darcher, Pedro J Gonçalves, Rachel Rapp, Ismail Elezi, Johannes Niediek, et al. Decoding movie content from neuronal population activity in the human medial temporal lobe. *bioRxiv : the preprint server for biology*, 2024. doi: 10.1101/2024.06.13.598791.

Joshua I. Glaser, Ari S. Benjamin, Raeed H. Chowdhury, Matthew G. Perich, Lee E. Miller, and Konrad P. Kording. Machine learning for neural decoding. *eNeuro*, 7(4), 2020. doi: 10.1523/ENEURO.0506-19. 2020. URL `https://www.eneuro.org/content/7/4/ENEURO.0506-19.2020`.

Uri Hasson, Rafael Malach, and David J Heeger. Reliability of cortical activity during natural stimulation. *Trends Cogn Sci*, 14 (1):40–48, December 2009.

Umit Keles, Julien Dubois, Kevin J M Le, J Michael Tyszka, David A Kahn, Chrystal M Reed, Jeffrey M Chung, Adam N Mamelak, Ralph Adolphs, and Ueli Rutishauser. Multimodal single-neuron, intracranial EEG, and fMRI brain responses during movie watching in human patients. *Sci. Data*, 11(1): 214, February 2024.

Gabriel Kreiman, Christof Koch, and Itzhak Fried. Category-specific visual responses of single neurons in the human medial temporal lobe. *Nature Neuroscience*, 3(9):946–953, 2000. doi: 10.1038/78868.

Fangyu Liu, Saber Meamardoost, Rudiyanto Gunawan, Takaki Komiyama, Claudia Mewes, Ying Zhang, EunJung Hwang, and Linbing Wang. Deep learning for neural decoding in motor cortex. *Journal of Neural Engineering*, 19. doi: 10.1088/1741-2552/ac8fb5.

OpenAI, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, et al. GPT-4o System Card, October 2024.

Rodrigo Quian Quiroga, Leila Reddy, Gabriel Kreiman, Christof Koch, and Itzhak Fried. Invariant visual representation by single neurons in the human brain. *Nature*, 435(7045):1102–1107, 2005. doi: 10.1038/nature03687.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models, 2024. URL `https://arxiv.org/abs/2401.00368`.

Yuwen Zhang, Zahra M. Aghajan, Matias J. Ison, et al. Decoding of human identity by computer vision and neuronal vision. *Scientific Reports*, 13(1):651, 2023. doi: 10.1038/s41598-022-26946-w.

9. Appendix

9.1. GPT Feature Extraction Details

Below is the full prompt used for GPT-based feature extraction. GPT returns a JSON object which is then parsed and

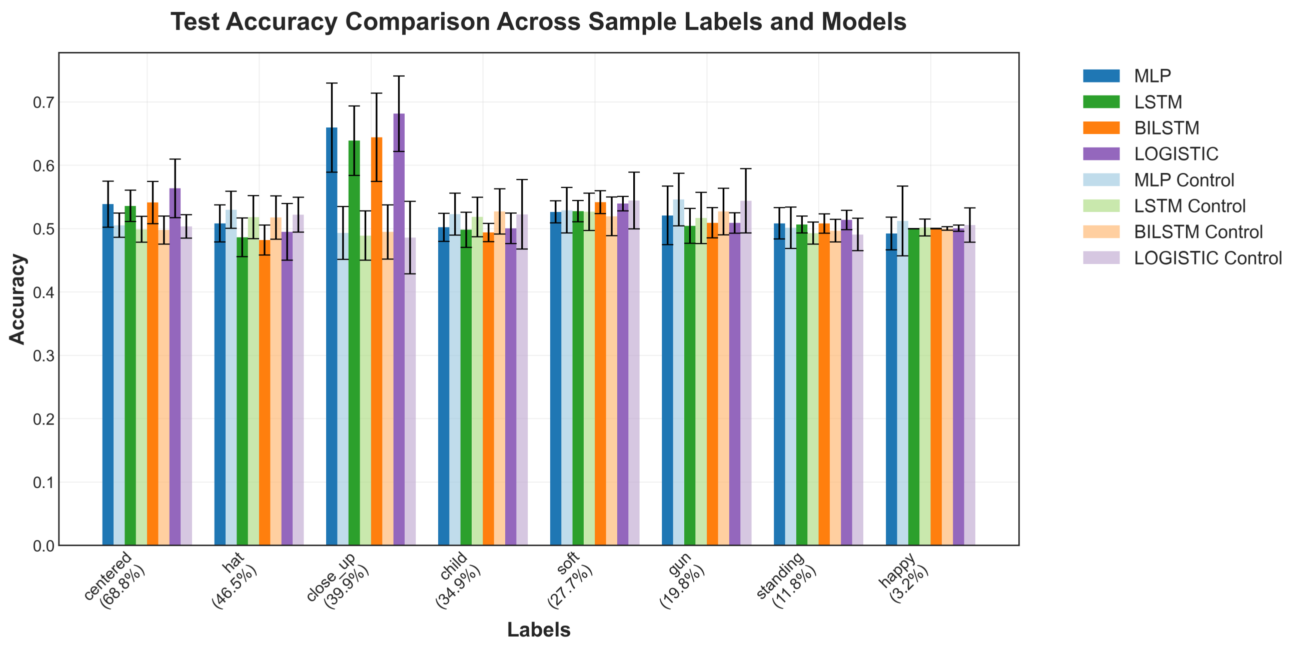

Figure 2: Model accuracy comparison across various models and labels. Only a subset of the GPT labels are shown here for ease of visualization. Our framework allows for the automatic search through a wide range of movie features, and accurately identifies the neurally decodable features (*close-up* and *centered* here).

stored in HDF5 format.

> Describe the visual elements in this frame objectively. Focus on:
>
> 1. Visible objects
>
> 2. Observable actions or movements
>
> 3. Emotional indicators including:
>
> - Facial expressions
>
> - Body language
>
> - Character interactions
>
> - Scene mood and atmosphere
>
> - Any visible emotional reactions
>
> Format your response as a valid JSON object with keys for 'objects', 'actions', and 'emotions'. Each key should have an array of strings as its value. Do not include any text outside of the JSON object.
>
> Try to be as objective as possible.
>
> For emotions, use only single-word labels, such as the following: joy, sadness, anger, fear, surprise, disgust, neutral, calm, excited, tense, relaxed. You can use multiple labels if appropriate, but stick to one-word labels.
>
> Important: If there are no people visible in the frame or no clear emotional indicators, leave the 'emotions' array empty.

The model returns a JSON object with three primary keys:

```
{
"objects": ["object1", "object2", ...],
"actions": ["action1", "action2", ...],
"emotions": ["emotion1", "emotion2", ...]
}
```

In addition, here is another prompt that we used for more detailed features:

> Analyze this frame with the detail and nuance of a professional film critic, including narrative context and audience impact. Provide a comprehensive analysis of:
>
> 1. Visual Composition:
>
> - Framing and camera angles
>
> - Lighting and shadow patterns
>
> - Depth and spatial relationships
>
> - Set design and environmental elements
>
> 2. Character and Action Analysis:
>
> - Detailed character descriptions (appearance, costumes, positioning)
>
> - Precise movements and gestures
>
> - Character interactions and power dynamics
>
> - Background action and ambient movement

- Props and their symbolic significance

3. Emotional and Atmospheric Elements:

- Detailed facial expressions
- Body language and posture
- Character emotional states
- Scene mood and tension
- Interpersonal dynamics
- Symbolic or metaphorical elements

4. Technical Elements:

- Focus and depth of field
- Movement patterns
- Visual effects or notable post-processing
- Frame composition techniques

5. Narrative Context:

- Plot progression indicators
- Scene purpose and dramatic function
- Character development moments
- Story beats and turning points
- Foreshadowing elements
- Scene position in narrative arc
- Relationship to previous/future scenes

6. Audience Experience:

- Expected emotional responses
- Tension and release patterns
- Viewer engagement elements
- Psychological impact
- Key takeaways or realizations
- Memorable or impactful moments

Format your response as a valid JSON object with the following structure:

```
{
    "visual_composition": {
        "framing": [],
        "lighting": [],
        "spatial_elements": []
    },
    "character_elements": {
        "descriptions": [],
        "actions": [],
        "interactions": [],
        "props": []
    },
    "emotional_atmosphere": {
        "expressions": [],
        "body_language": [],
        "mood_indicators": [],
        "emotional_states": []
    },
    "technical_aspects": {
        "camera_techniques": [],
        "composition_notes": [],
        "notable_effects": []
    },
    "narrative_context": {
        "plot_progression": [],
        "scene_purpose": [],
        "character_arcs": [],
        "story_position": [],
        "foreshadowing": []
    },
    "audience_impact": {
        "emotional_responses": [],
        "engagement_elements": [],
        "psychological_effects": [],
        "key_takeaways": [],
        "memorable_moments": []
    }
}
```

Each array should contain detailed string descriptions. Be specific and thorough in your observations, using professional cinematographic and critical terminology where appropriate. Consider both the immediate frame content and its broader narrative context. If any element is not visible or applicable, provide an empty array for that category.

For narrative elements, analyze how this frame contributes to the larger story, even if it's a subtle moment. For audience impact, consider both immediate emotional responses and deeper psychological effects. Use understanding of storytelling techniques and viewer psychology to make informed interpretations.

In addition, we provide the following system prompt to GPT-4:

You are analyzing frames from the movie described below. Use this context to inform your analysis:

and then input the following summary loaded from a text file. We determine the scene numbers using a file provided by our mentors.

This is an 8 minute edited version of a television episode of Alfred Hitchcock Presents called

"Bang You're Dead." The original version is 20 minutes long. Please use the following description of the movie as context to help annotate the movie frames.

The episode is about a boy named Jacky who finds a real gun in his uncle's suitcase and thinks it is a toy gun. He loads in a bullet and goes around pointing the loaded gun at various characters while pulling the trigger. Viewers feel suspense and anxiety thinking that the gun will go off and kill someone.

The characters in the movie/video clip are:

Jacky: the young boy who carries a real loaded gun throughout the movie, pointing it at people, and is at risk of accidentally shooting someone. He wears a cowboy hat. Friend1: Jacky's friend with the striped shirt who shows up at the beginning of the movie with a toy gun that looks real. Friend2: Friend of Jacky and Friend1. He only appears in the beginning of the movie and is hiding behind a tree. Friend1 pretends to shoot Friend2 with the toy gun. Dad: Jacky's dad. He at first dismisses Jacky's behavior and then helps ensure that the maid is not shot. Mom: Jacky's mom. She initially believed Jacky had a toy gun but later realizes that Jacky had a real, loaded gun Uncle: Uncle Rick who has a surprise for Jacky. Jacky finds a real gun in Uncle's suitcase and mistakes it for his present. Jacky mistook the gun and bullets to be the present. StoreClerk: LittleGirl: She tries the mechanical horse ride that Jacky is riding and insists him to leave. LittleGirlsDad: Dad initially wants to say no to his daughter but eventually gives in and tricks Jacky into leaving. Jacky almost shoots him. Cleo: Maid who Jacky almost shoots. Other minor characters.

Scene 1:

Friend1 opens the movie with a toy gun that looks real. Friend1 and Jacky are hiding behind a tree having a pretend shootout with Friend2 who is hiding behind another tree. Jacky is wearing a cowboy hat. Friend1 is wearing a striped shirt and no hat. Friend1 pretends to shoot Friend2, who is hiding behind a tree. Friend1 has received a toy gun that looks real for Friend1's birthday. Jacky immediately shows interest in the gun. Jacky is interested in playing with Friend1's gun too, but Friend1 refuses.

Scene 2:

Jacky stands next to his Mom and Uncle. Uncle talks with Mom. Dad hands Uncle a drink.

Scene 3:

Uncle has a surprise for Jacky and tells him he will give it to him after the party.

Scene 4: Dad enters the room and hands uncle a drink. Jacky tells his dad Uncle has a surprise for him. Dad asks Jacky to unpack for Uncle. Door closes and Jacky is in the room by himself.

Scene 5: Jacky notices the Uncle's gun and starts to play with it. He assumes it is his present. Jacky takes out bullets and puts them into his pocket. He places one into the gun and rotates the revolver, just managing to have the bullet miss the gun's exit.

Scene 6: The Uncle speaks about a museum piece. Jacky pretends to shoot Mom, and she thinks he is joking. Mom has the strangest feeling that something is wrong. Dad and Mom ask him to play outside, and Jacky thanks the Uncle for the surprise.

Scene 7:

Jacky rides a mechanical horse, pretending to shoot a stranger. He adds more bullets to the gun.

Scene 8: Mom realizes Jacky has a real gun. Jacky thinks the uncle's revolver he has is a toy.

Scene 9: Dad says he will call the police. Dad, Mom, and Uncle drive away to find Jacky.

Scene 10: Jacky pretends to shoot more people. LittleGirl jumps on the horse and wants to ride it. LittleGirlsDad initially wants to say no to her, but he gives in and picks Jacky off the horse. He points the gun at LittleGirlsDad and rotates the revolver before shooting.

Scene 11: Jacky returns home.

Scene 12: Jacky steps inside and shuts the door behind him.

Scene 13: Jacky points the gun at Cleo, threatening to shoot her. Dad throws an object at Jacky to make him miss Cleo. He takes the gun from Jacky, and Jacky runs to his Mom shocked. Mom was relieved that Cleo wasn't hurt. Dad and Uncle look at each other in shock.

To ensure output quality and reliability, the pipeline includes:

- Validation of JSON structure before storage
- Automatic retry mechanisms for failed API calls
- Logging of parse errors and response anomalies
- Regular format verification during checkpointing

To ensure reliable and consistent feature extraction, we implemented several important constraints and guidelines in our prompting strategy. The prompt explicitly requests relatively objective descriptions in the persona of a movie critic, reducing potential biases in feature extraction. Emotional labels are restricted to single-word descriptors, preventing overly nuanced emotional attributions that could complicate analysis. The prompt includes explicit instructions to leave the information empty when no clear indicators are present, reducing false positives and ensuring annotation reliability allowing for thorough responses to allow for flexible one-hot encodings.

Our feature extraction pipeline is implemented as a modular Python framework that handles the entire process from video processing to structured data storage. At its core, the pipeline utilizes OpenCV (cv2) for video frame processing, GPT-4 Vision within the GPT-4o API for feature extraction, and HDF5 for efficient data storage and retrieval. The pipeline processes the video frame by frame. Each sampled frame is saved as a JPEG image and referenced in the HDF5 output, maintaining a clear link between extracted features and source frames. Each frame is encoded as a base64 string and sent alongside our carefully crafted prompt. The API call is configured with a variable token limit to ensure concise, focused responses while maintaining comprehensive feature coverage. Our first prompt used a limit of 300 tokens, while our (much longer) second prompt used a limit of 1000 tokens.

The response processing pipeline includes layers of validation and error handling. It starts by checking API status codes for errors, then removes Markdown artifacts like JSON code block markers using regular expressions. The cleaned content is parsed as JSON, with unparseable responses saved as raw content for later review.

Data is stored in a hierarchical HDF5 structure, with each frame's data in a separate group labeled by frame number. Attributes store filenames, and datasets hold extracted features (objects, actions, emotions). Unicode strings are converted to UTF-8 bytes, and JSON serialization is used for lists. To maintain integrity during processing, the pipeline checkpoints extracted features every 100 frames.

9.2. Neural Data and Experimental Setup

As multiple models of different architectures were used for decoding, standardization was a critical step to ensure comparability across models.

9.2.1. Temporal Windows

To aid comparison with LSTMs, we employ a "temporal window" approach for the MLP and logistic models. Each window represents how many previous frames are included as input to the model when making a prediction. Specifically, for each frame i, we include the previous $(\textit{window} - 1)$ frames as input to all three models. We used temporal window sizes of $1, 5, 10, 20$ for analysis.

9.2.2. Temporal Blocking

In movies, consecutive frames within a shot exhibit high temporal correlation (or more simply, they are visually very similar). For example, in "Bang, You're Dead," when a particular angle of a gun is displayed on scene for 2 seconds, many frames identically contain the gun in the same position. Therefore, the particular train/val/test split used could drastically affect the model results and lead to an inaccurate evaluation of the model accuracy. Traditional random splitting can result in data leakage when similar frames end up in both training and testing sets, artificially inflating performance metrics. Therefore, we implement the following temporal blocking strategy, splitting frames into 70% training, 15% validation, and 15% test sets:

Continuous blocks of frames are kept together during splitting, maintaining temporal coherence while reducing data leakage. We experimented with various block sizes (1, 10, 50, 100, 200, 500, 1000, 2000 frames) to analyze the tradeoff between temporal separation and model performance. The data is first divided into blocks of the specified size, then these blocks are randomly assigned to train, validation, and test sets. This ensures that frames within the same temporal block remain together.

We aim to identify the block size that can reduce data leakage between splits without being so aggressive that some labels are almost entirely in one split. To determine such a block size, we train a logistic regression model on the human binary labels, taking the average and standard deviation of the percent correct (accuracy) metric over 1000 different split seeds. As shown in Figure 2, accuracy plateaus after a block size of 1000, indicating that this block size is ideal for preventing data leakage while not being too large. We note that we say similar results with the MLP, corroborating our choice of 1000 for the block size.

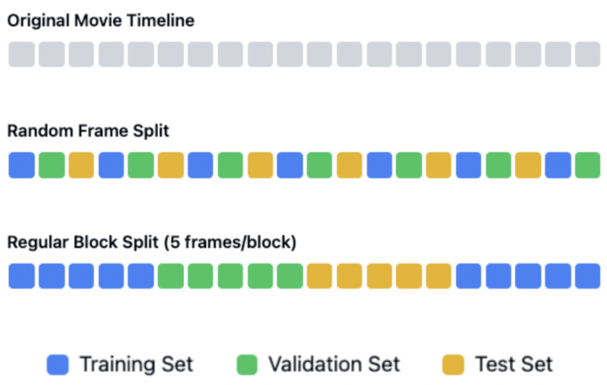

Figure 3: Visualization of random splitting compared to blocking. Random splitting assigns individual frames to sets independently, while temporal blocking maintains fixed-size blocks of consecutive frames.

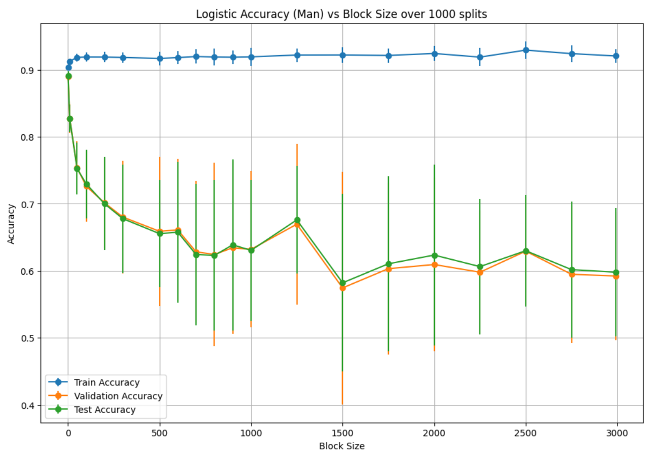

Figure 4: Logistic model accuracy vs. block size for man decoding, over 1000 seeds used for randomly assigning train/val/test to blocks. Note the train accuracy remaining the same over block sizes (overfitting), but the sharp decrease in val/test accuracy as block size initially increases (data leakage for small block size) and eventual plateauing (lack of sufficient number of labels in split for large block size). We choose 1000 as our ideal block size, preventing data leakage while ensuring some degree of randomness and sufficient number of labels in split.

9.2.3. Balancing

The percentage of positive labels varies widely across the word label features. As such, we balance by oversampling the minority class:

1. **Dataset Loading and Identification of Minority Class**
   Let $D = \{(x_i, y_i)\}_{i=1}^N$ be the original dataset, where $y_i \in \{0, 1\}$.
   Identify the minority class:
   $$y_{\text{minority}} = \arg\min_y |\{i \mid y_i = y\}|$$

2. **Assigning Blocks to Training, Validation, and Test Sets**
   Randomly assign each block B_j to one of the subsets Train, Val, or Test based on predefined split ratios (e.g., 70% Train, 15% Val, 15% Test). Ensure that the block indices are temporally ordered to maintain chronological integrity.

3. **Creating Copies for Balancing**
   For each subset $S \in \{\text{Train}, \text{Val}, \text{Test}\}$, create two copies:
   $$S_{\text{original}} \quad \text{and} \quad S_{\text{balanced}}$$

4. **Balancing the Balanced Copies via Oversampling**

   4.1. **Oversampling Without Temporal Windows**
   For each balanced subset S_{balanced}, apply oversampling to the minority class y_{minority} such that:
   $$|y_{\text{minority}}| = |y_{\text{majority}}|$$
   This is achieved by replicating blocks from the minority class until the number of minority samples equals that of the majority class.

   4.2. **Oversampling with Temporal Windows**
   If using temporal windows, ensure that S_{balanced} contains sequences:
   $$\{(x_i, \dots, x_{i+T-1}, y_{i+T-1})\}$$
   Instead of single vectors (x_i, y_i).
   Oversample sequences where:
   $$y_{i+T-1} = y_{\text{minority}}$$
   to balance the dataset at the sequence level.

5. **Finalizing the Balanced and Original Datasets**
   Define the original and balanced datasets as:
   $$D_{\text{original}} = \{\text{Train}_{\text{original}}, \text{Val}_{\text{original}}, \text{Test}_{\text{original}}\}$$
   $$D_{\text{balanced}} = \{\text{Train}_{\text{balanced}}, \text{Val}_{\text{balanced}}, \text{Test}_{\text{balanced}}\}$$

9.2.4. Metrics

We employ a variety of metrics to account for the unbalanced feature labels. Although we evaluate all metrics on the train, val, and test sets, we make two test sets: one that is balanced and one that is unbalanced. For the balanced test set, we compute

- **Accuracy**: the proportion of correctly predicted labels out of all predictions. This provides a straightforward evaluation of overall model performance when the test data is balanced.
- **Balanced Accuracy:** Computes the average recall for all classes, ensuring that performance is measured fairly even when class distributions are unequal.

For the unbalanced test set, we compute

- **Cohen's Kappa:** Evaluates the agreement between predicted and actual labels, accounting for chance agreement. Useful for imbalanced datasets where class proportions may skew simpler metrics.
- **F1 Score:** The harmonic mean of precision and recall, balancing false positives and false negatives. It is particularly effective for datasets with imbalanced classes.
- **AUROC (Area Under the Receiver Operating Characteristic Curve):** Measures the model's ability to distinguish between classes across all thresholds. Higher values indicate better discriminatory power.
- **PR AUC (Precision-Recall Area Under Curve):** Focuses on the balance between precision and recall for different thresholds, especially valuable for imbalanced datasets where the positive class is of greater interest.

9.2.5. K-Seed Cross Validation

To ensure robust evaluation of our models' performance, we implement a k-seed cross validation strategy with k=3. This approach provides a more reliable assessment of model generalization than single-split validation while accounting for the temporal nature of our data. For each model configuration and feature type, we generate three distinct random seeds (k=3) for creating different temporal block splits. For each seed, we create new train/val/test splits while maintaining temporal block structure. Then, we train and evaluate the model independently for each seed, aggregating results across all seeds to compute mean performance metrics and standard deviations. This process helps account for variance in model performance due to different temporal block arrangements.

The validation strategy can be formalized as:

$$\text{Metrics}_{\text{final}} = \frac{1}{k}\sum_{i=1}^{k}\text{Metrics}(\text{Model}_i) \tag{1}$$

where Model_i represents the model trained with the i-th seed's data split, and $\text{Metrics}(\cdot)$ represents our evaluation metrics (accuracy, F1 score, etc.). This approach provides more stable performance estimates while capturing the variance introduced by different temporal block arrangements.

For our experiments, we set k=3 as it provides a good balance between computational cost and statistical reliability.

9.2.6. Statistical Validation Through Controls

To validate our decoding results, we implement a control framework that combines k-seed cross validation with multiple control shuffles. For each of our three cross-validation seeds, we perform two independent control shuffles of the binary labels, resulting in a total of nine experimental runs per word label feature: three for the actual decoding and six for controls. This design enables statistical comparison between real and chance-level performance.

The primary control condition implements a temporal shift in the labels while maintaining the temporal structure of neural data. Specifically, for each control shuffle, we generate a random shift value between 2,000 and 10,000 frames and circularly shift the binary labels by this random amount using:

$$y_{\text{control}}[i] = y_{\text{original}}[(i + \text{shift}) \bmod N]$$

where N is the total number of frames. We then maintain the original temporal structure of neural firing rates.

This approach preserves the temporal autocorrelation structure of both the neural data and the labels while breaking their temporal relationship, providing a more stringent control than random shuffling.

For each feature, we obtain three sets of metrics from the original data (one per CV seed) and six sets from the controls (two shuffles per CV seed). We then employ a one-sided Welch's t-test to compare these distributions, accounting for potentially unequal variances:

$$t = \frac{\bar{X}_r - \bar{X}_c}{\sqrt{\frac{s_r^2}{n_r} + \frac{s_c^2}{n_c}}} \tag{2}$$

where $\bar{X}_r, s_r^2$ are the mean and variance of the real decoding performance across seeds, and $\bar{X}_c, s_c^2$ are the mean and variance of the control performance. The variance of the control data s_c^2 is approximated by taking the sum of the variances between control shuffles and between blocking splits:

$$s_c^2 = \text{Var}\left(\{\overline{x}_s\}_{s=1}^{S}\right) + \frac{1}{S}\sum_{s=1}^{S}\text{Var}\left(\{x_{s,b}\}_{b=1}^{B}\right) \tag{3}$$

where:

S is the total number of control shuffles,

B is the number of splits in each shuffle,

$\overline{x}_s$ is the average metric for shuffle s,

$x_{s,b}$ is the metric for shuffle s and split b,

Var denotes variance.

This statistical framework allows us to account for variance in both real and control performance, handle unequal sample sizes between real (n=3) and control (n=6) results, and provide rigorous p-values for assessing decoding significance.

The combination of k-seed cross validation with multiple control shuffles per seed provides a robust framework for validating our decoding results against chance-level performance while accounting for the temporal structure of the data.

9.3. Results from other metrics

From the features extracted using the ChatGPT feature extraction pipeline, we run binary classification for all the models, MLP, Logistic, LSTM and BiLSTM using our standardized process with balancing, cross validation and evaluation methods detailed above.

From the accuracy plot below and associated p-values, we find that man and close-up are consistenly significantly decodable across models, performing better than controls, indicating the model is detecting and decoding relevant neural signals for these labels. Additionally, centered and standing are also decodable, but only for logistic and LSTM and BiLSTM with certain temporal window sizes. We later observe that some of the metrics have skewed results such as AUROC and we add this to the appendix for future inspection. We hypothesize that our balencing strategies for each of the decoded labels is not the most optimal which causes these metrics to be skewed. Future work could include an in-depth exploration of the effects of balencing on different metrics.

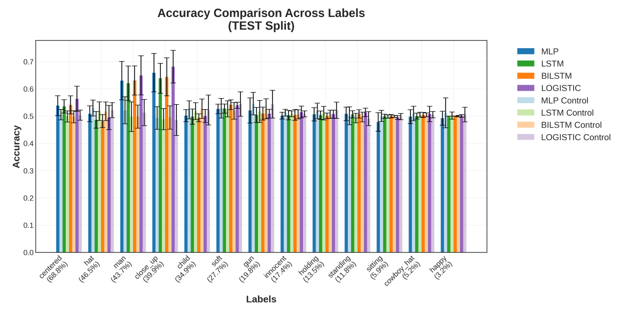

Figure 5: Comparison of the metric accuracy across all models across all labels with controls for each model type on Test data

Figure 6 shows that for the labels that are decodable (man, close-up) and centered/standing for specific temporal window sizes and specific models, the AUROC score is generally above 0.5, indicating the decoding is not happening by chance. However, compared to the accuracy plot (5) it seems that the controls generally perform similar to the original datasets with regards to AUROC. This is because AUROC is based on relative ranking of predictions instead of the actual prediction values, so even if certain labels are decodable, the label shift causes a similar ranking of predictions.

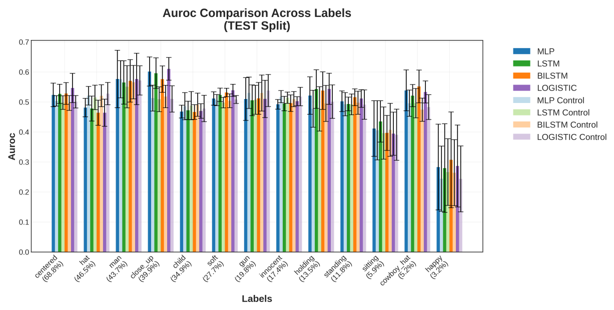

Figure 6: Comparison of the metric AUROC across all models across all labels with controls for each model type on Test data

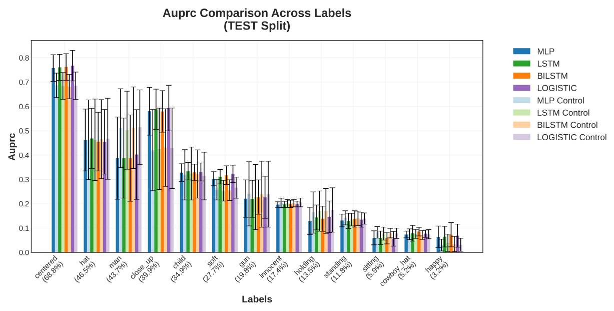

Figure 7: Comparison of the metric AUPRC across all models across all labels with controls for each model type on Test data

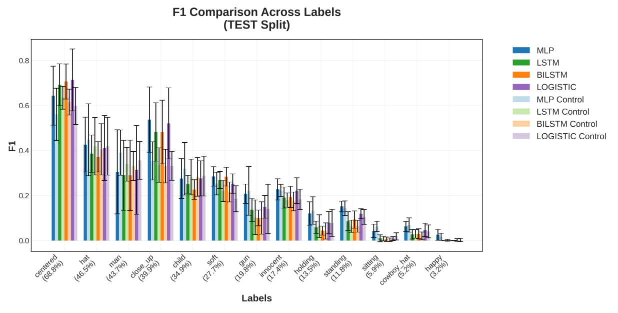

Figure 8: Comparison of the metric F1 across all models across all labels with controls for each model type on Test data

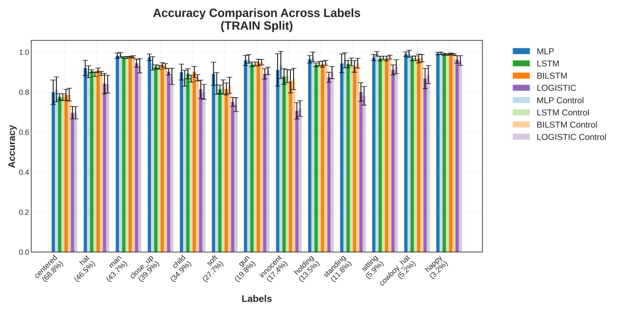

Figure 9: Comparison of the metric Accuracy across all models across all labels with controls for each model type on Train data

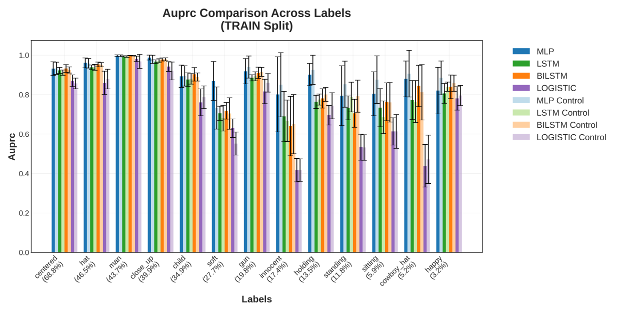

Figure 10: Comparison of the metric AURPC across all models across all labels with controls for each model type on Train data

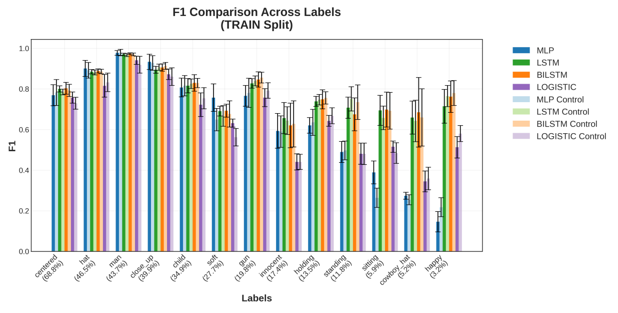

Figure 11: Comparison of the metric F1 across all models across all labels with controls for each model type on Train data

5-8 show results across all models and labels, where labels are ordered by frequency. From the accuracy plot below (5) and associated p-values, we find that "man" and "close-up" are consistently significantly decodable across models, performing better than controls. This indicates that the model is detecting and decoding relevant neural signals for these labels. Additionally, "centered" and "standing" are also decodable, but only for Logistic, LSTM, and BiLSTM with certain temporal window sizes.

Overall, the analysis reveals that while certain labels like "man" and "close-up" are universally decodable, others like "gun" and "centered" rely on specific models or temporal information.

9.4. Model Architectures and Implementation

The training procedure uses binary cross-entropy with logits loss and the Adam optimizer for all models. We implemented all models with PyTorch Lightning.

Multi-Layer Perceptron (MLP) The MLP architecture consists of a four-layer network. The input layer accepts population firing rates from recorded neurons, followed by three hidden layers with progressively decreasing sizes and a single output neuron with sigmoid activation for binary classification. Each hidden layer employs ReLU activation and includes dropout (rate=0.2) for regularization.

Training parameters for the MLP were optimized through preliminary experiments, leading to the following configuration:

- Hidden layer size: 64 units
- Batch size: 256
- Number of epochs: 100
- Dropout rate: 0.2
- Learning rate: 0.0001

Logistic Regression The logistic model serves as our baseline model for binary classification. A high regularization constant of 10,000 was used as it was found that this leads to the least overfitting and highest validation accuracy.

LSTM (Long Short-Term Memory) The LSTM architecture implements a sequence modeling approach to capture temporal dependencies in neural firing patterns. The model consists of multiple stacked LSTM layers with the following structure:

- **Input Representation**: The input layer accepts sequences of neural population firing rates with shape (batch_size, sequence_length, input_dimension), where input_dimension corresponds to the number of recorded neurons.
- **LSTM Layers**: Two stacked LSTM layers process the temporal sequences, each with hidden_dimension units. These layers maintain internal cell states and hidden states, allowing the model to learn both short-term and long-term dependencies in the firing patterns. A dropout rate of 0.2 is applied between layers for regularization.
- **Output Layer**: The final hidden state from the last LSTM layer is passed through a fully connected layer to produce the classification output.

The LSTM's ability to selectively remember or forget information through its gating mechanisms makes it particularly suitable for capturing temporal patterns in neural activity that might extend beyond immediate time windows.

BiLSTM with Attention The Bidirectional LSTM with Attention mechanism extends the basic LSTM architecture to capture more complex temporal relationships:

- **Bidirectional Processing**: The input sequence is processed in both forward and backward directions using two separate LSTM layers, each with hidden_dimension units. This bidirectional approach allows the model to capture dependencies from both past and future time steps at each point in the sequence.
- **Attention Mechanism**: An attention layer is implemented on top of the BiLSTM outputs, computing attention weights for each time step. This mechanism allows the model to dynamically focus on relevant parts of the input sequence when making predictions. The attention weights are learned during training and provide interpretable importance scores for different time points in the neural activity.
- **Output Processing**: The attention-weighted context vector is passed through two dense layers with dropout (0.2) for final classification. The first dense layer uses ReLU activation and reduces dimensionality, while the final layer produces the classification output.

The combination of bidirectional processing and attention mechanisms allows this architecture to capture more nuanced temporal relationships in the neural data, potentially identifying key moments in the firing patterns that are most relevant for predicting the target variables.

For binary classification tasks, we employ grid search over key hyperparameters: learning rates [1e-3, 1e-4], hidden dimensions [32, 64], and number of layers [1, 2]. The BiLSTM architecture concatenates forward and backward hidden states, effectively doubling the model's capacity to capture temporal dependencies at the cost of increased computational complexity.

9.5. Model-Wise P-Values by Feature

Model	Centered	Hat	Close-Up	Child
MLP	*0.0985*	0.8160	**0.0398**	0.9737
LSTM	*0.0707*	0.9342	**0.0338**	0.9790
BiLSTM	**0.0457**	0.8414	**0.0281**	0.9897
Logistic	*0.0698*	0.7304	**0.0090**	0.7785

Model	Soft	Gun	Standing	Happy
MLP	0.5390	0.5614	0.3405	0.7461
LSTM	0.5631	0.6172	*0.0827*	0.7646
BiLSTM	0.1557	0.6010	*0.0650*	0.5939
Logistic	0.7913	0.8022	*0.0674*	0.7083

Table 1: P-values across model types (left) and extracted feature labels (top). Bold indicates $p < 0.05$. Italics indicates $p < 0.1$.