Module Interface Specification for Model–to–Brain Alignment for Speech Recognition

Xiao Shao

April 22, 2026

1 Revision History

Date	Version	Notes
March 20, 2026	1.0	Initial version
April 21	1.1	Update according to smith's feedback

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at SRS.tex

Contents

3 Introduction

The following document details the Module Interface Specifications for Model–to–Brain Alignment for Speech Recognition. This pipeline accepts different deep learning-based models for MEG analyzing, performing mTRF test for comparison.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/Krozix-2026/cas741-speech-trf. The key companion documents are SRS and MG.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995).

The following table summarizes the primitive data types used by Model–to–Brain Alignment for Speech Recognition.

Symbol	Unit	Description
C	–	Number of MEG sensors.
D	–	Predictor feature dimension.
D_b	–	Feature dimension of the baseline predictor matrix $\mathbf{R}^{(\text{base})}$.
B	–	Training batch size.
E	–	Number of training epochs.
K	–	Number of time lags used in the TRF model.
L	–	Number of candidate predictors or layers being compared.
N	–	Number of aligned time samples used for TRF fitting.
N_{ROI}	–	Number of regions of interest (ROI), if ROI-based analysis is performed.
$\mathcal{D}$	–	Speech dataset collection.
$\mathcal{D}_{\text{train}}$	–	Training split of dataset $\mathcal{D}$.
$\mathcal{D}_{\text{val}}$	–	Validation split of dataset $\mathcal{D}$.
$\mathcal{D}_{\text{test}}$	–	Test split of dataset $\mathcal{D}$.
$\boldsymbol{a}$	–	Audio input sequence provided to the speech model.
$\boldsymbol{s}^{\star}$	–	Ground-truth transcript for ASR training.
$\boldsymbol{\theta}$	–	Trainable parameters of the speech model.
$\boldsymbol{\theta}^{*}$	–	Optimized model parameters after training.
$f_{\boldsymbol{\theta}}(\cdot)$	–	Speech model parameterized by $\boldsymbol{\theta}$.

Continued on next page

Symbol	Unit	Description
$\mathcal{L}_{\text{ASR}}$	–	ASR training loss used to fit $f_{\boldsymbol{\theta}}$.
η	–	Learning rate for gradient-based optimization.
ℓ	–	Layer index of the deep speech model.
H_ℓ	–	Hidden-state dimensionality of layer ℓ.
n	–	Discrete time index after temporal alignment or model frame sampling.
t	s	Continuous time variable in seconds.
Δt	s	Sampling interval of MEG signals, $\Delta t = 1/f_s$.
$\boldsymbol{h}_n^{(\ell)}$	–	Hidden state vector from layer ℓ at discrete time index n.
$g(\cdot)$	–	Optional transformation applied to hidden states to form predictors.
$\boldsymbol{r}_n^{(\ell)}$	–	Predictor representation vector derived from $\boldsymbol{h}_n^{(\ell)}$.
$\mathbf{R}^{(\ell)}$	–	Time-by-feature predictor matrix derived from layer ℓ, with $\mathbf{R}^{(\ell)} \in \mathbb{R}^{N \times D}$.
$\mathbf{R}^{(\text{base})}$	–	Baseline predictor matrix, with $\mathbf{R}^{(\text{base})} \in \mathbb{R}^{N \times D_b}$.
$\mathbf{X}(\mathbf{R})$	–	Lagged design matrix constructed from predictor matrix $\mathbf{R}$, with $\mathbf{X}(\mathbf{R}) \in \mathbb{R}^{N \times (KD)}$.
$\mathbf{X}$	–	Design matrix formed from time-lagged predictors when the underlying predictor matrix is clear from context.
$\mathbf{X}_\tau$	–	Portion of the design matrix corresponding to lag τ.
f_s	Hz	Sampling frequency of MEG signals used for TRF modeling.
f_a	Hz	Sampling frequency of the raw audio waveform.
τ	s	Time-lag variable in the TRF model.
$\tau_{\min}$	s	Start of the TRF lag window.
$\tau_{\max}$	s	End of the TRF lag window.
$\boldsymbol{x}(n)$	–	Predictor vector at aligned discrete time index n.
$\boldsymbol{y}(n)$	T / fT	MEG response vector across sensors at aligned discrete time index n.
$\mathbf{Y}$	T / fT	MEG response matrix across time and sensors.
$\hat{\mathbf{Y}}$	T / fT	Predicted MEG responses from the fitted encoding model.
$\mathbf{w}(\tau)$	–	TRF weight vector at lag τ.
$\mathbf{W}$	–	Full TRF weight matrix across lags and sensors.
$\mathbf{W}^*$	–	Ridge-regularized estimate of the TRF weights.
λ	–	Ridge regularization coefficient for TRF fitting.
ρ	–	Prediction score of an encoding model.

Continued on next page

Symbol	Unit	Description
ρ_{base}	–	Encoding score obtained using only the baseline predictor.
ρ_ℓ	–	Encoding score obtained using predictors derived from layer ℓ.
$\Delta\rho_\ell$	–	Improvement over baseline, defined as $\Delta\rho_\ell = \rho_\ell - \rho_{\text{base}}$.
ρ_{CV}	–	Cross-validated prediction score.

The specification of Model–to–Brain Alignment for Speech Recognition uses some derived data types: sequences, strings, tuples, matrices, and tensors. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. Matrices and tensors denote multi-dimensional arrays of real values. In addition, Model–to–Brain Alignment for Speech Recognition pipeline uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	Level 2
Hardware-Hiding	Audio Data Module
	MEG Data Module
Behaviour-Hiding	Input Parameters Module
	Model Training Module
	Frozen Model Module
	Representation Extraction Module
	Statistical Analysis Module
Software Decision	Plotting Module

Table 2: Module Hierarchy

6 MIS of Audio Data Module

6.1 Module

Audio Data Module

6.2 Uses

Input Parameters Module

6.3 Syntax

6.3.1 Exported Constants

- $\mathcal{S}_{\text{Audio}}$: the finite set of supported audio dataset identifiers. In the current implementation this includes at least the LibriSpeech-based training data used by the speech-model training pipeline.
- $\mathcal{F}_{\text{Seq}}$: the abstract type of a time-by-feature matrix, represented in the current implementation as a tensor of shape (T, F).
- $\mathcal{B}_{\text{Audio}}$: the abstract type of a batched audio training sample. In the current implementation this is represented by `LibriSpeechBatch`, `AlignedWordBatch`, or an equivalent dictionary-based batch record, depending on the training task.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
LibriSpeechASR	root: Path, subset: string, vocab: CharVocab, sample_rate: int, n_mels: int, win_length_ms: real, hop_length_ms: real, download: bool, limit_samples: optional int	dataset object	dataset unavailable, metadata unavailable, file error
LibriSpeechASR.			
__getitem__	idx: int	record containing feat, feat_len, target, target_len, text, utt_id	index out of range, file error, malformed audio
LibriSpeechAlignedWords	manifest_path: Path, subset: string, vocab: WordVocab, tail_frames: int, limit_samples: optional int	dataset object	manifest unavailable, malformed manifest, feature file error
LibriSpeechAlignedWords.			
__getitem__	idx: int	record containing feat, feat_len, target, word_ids, word_starts, word_ends, utt_id	index out of range, malformed feature array, file error
buildWordVocabFrom Manifest	manifest_path: Path, topk: int	WordVocab	manifest unavailable, malformed manifest
collateLibriSpeech	sequence of LibriSpeechASR item records	LibriSpeechBatch	malformed item record, inconsistent feature dimension
collateRNNT	sequence of LibriSpeechASR item records	dictionary batch record	malformed item record, inconsistent feature dimension
collateAlignedWords	sequence of LibriSpeechAlignedWords item records	AlignedWordBatch	malformed item record, inconsistent feature dimension

6.4 Semantics

6.4.1 State Variables

- *subset*: string, the dataset subset currently represented by a dataset object (for example, train, validation, or test subset).
- *indices*: optional sequence of integers, the restricted sample indices used when `limit_samples` is specified.
- *sampleRate*: positive integer, the target sampling rate used for audio loading and optional resampling.
- $featureSpec$: record containing the feature-extraction parameters used by the audio dataset object, including *n_mels*, *win_length_ms*, and *hop_length_ms*.
- $manifestItems$: sequence of manifest records used by `LibriSpeechAlignedWords`. Each manifest record contains at least the subset identifier, the cochlear-feature path, the word alignment list, and the utterance identifier.
- *vocab*: the vocabulary object used to encode targets. Depending on the task, this is either a character vocabulary or a word vocabulary.

6.4.2 Environment Variables

- Audio files, feature files, and manifest files stored in the external file system.

6.4.3 Assumptions

- The dataset paths and manifest paths supplied by the Input Parameters Module are valid.
- For `LibriSpeechASR`, each referenced audio file can be decoded into a waveform and, if necessary, resampled to the requested target sampling rate.
- For `LibriSpeechAlignedWords`, each manifest record contains a valid `coch_path` whose array has shape $(64, T)$.
- Each utterance identifier is unique within the dataset subset being used.

6.4.4 Access Routine Semantics

LibriSpeechASR(root, subset, vocab, ...):

- transition: initializes a dataset object backed by `torchaudio.datasets.LIBRISPEECH`, stores the requested subset and feature-extraction parameters, and prepares the mel-spectrogram and amplitude-to-dB transforms.

- exception: raised if the requested subset cannot be loaded or if the dataset root is invalid.

LibriSpeechASR.__getitem__(idx):

- output: uses dataset metadata to locate the corresponding audio file, loads the waveform through the local audio-loading routine, optionally resamples it to the configured sampling rate, computes the log-mel feature matrix of shape (T, F), normalizes the transcript text, encodes the target sequence, and returns a record containing the feature matrix, feature length, target sequence, target length, normalized text, and utterance identifier.

- exception: raised if the index is invalid, the audio file cannot be read, or the waveform cannot be transformed into the required feature format.

LibriSpeechAlignedWords(manifest_path, subset, vocab, ...):

- transition: loads the manifest records for the requested subset, stores the tail-frame setting and the vocabulary object, and prepares the aligned-word dataset object.

- exception: raised if the manifest file is unavailable or malformed.

LibriSpeechAlignedWords.__getitem__(idx):

- output: loads the cochlear feature array referenced by the selected manifest record, converts it into a time-by-feature tensor, constructs the frame-level target sequence together with the word-id, word-start, and word-end sequences, and returns the corresponding item record.

- exception: raised if the index is invalid, the feature file is missing, or the loaded feature array has an unexpected shape.

buildWordVocabFromManifest(manifest_path, topk):

- output: scans the manifest file, counts word occurrences, and returns a vocabulary object containing the most frequent `topk` word types together with an unknown-token identifier.

- exception: raised if the manifest file cannot be read or contains malformed records.

collateLibriSpeech(items):

- output: sorts item records by feature length, pads the feature tensors to a common time dimension, concatenates the target sequences, and returns a `LibriSpeechBatch`.

- exception: raised if the item records are malformed or have inconsistent feature dimensions.

collateRNNT(items):

- output: sorts item records by feature length, pads feature tensors and target tensors separately, and returns a dictionary-based batch record for RNNT-style training.
- exception: raised if the item records are malformed or have inconsistent feature dimensions.

collateAlignedWords(items):

- output: sorts aligned-word item records by feature length, pads the feature tensors, target tensors, and word-boundary arrays, and returns an `AlignedWordBatch`.
- exception: raised if the item records are malformed or have inconsistent feature dimensions.

6.4.5 Local Functions

- **_load_audio_soundfile**: loads an audio waveform from disk and converts stereo input to mono when needed.
- **normalize_text**: converts transcripts into the normalized text form used for target encoding.
- **feature extraction transform**: converts waveforms into log-mel features using the configured mel-spectrogram and amplitude-to-dB transforms.

7 MIS of MEG Data Module

7.1 Module

MEG Data Module

7.2 Uses

Input Parameters Module

7.3 Syntax

7.3.1 Exported Constants

- $\mathcal{D}_{\text{MEG}}$: the collection of MEG recordings and associated metadata available to Model–to–Brain Alignment for Speech Recognition.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
initMEGData	-	-	unsupported dataset, file error
getSubjectList	-	sequence of strings	dataset not initialized
getMEGResponse	subject id: string, utterance id: string	$\mathbf{Y}$	subject not found, trial not found, file error
getSensorCount	-	C	dataset not initialized
getMEGSamplingRate		f_s	dataset not initialized
getTrialMetadata	subject id: string, utterance id: string	tuple	subject not found, trial not found
getStimulusLink	subject id: string, utterance id: string	string	subject not found, trial not found

7.4 Semantics

7.4.1 State Variables

- $subjectIndex$: a mapping from subject identifiers to MEG recording descriptors.
- $trialIndex$: a mapping from ($subject\ id$, $utterance\ id$) pairs to dataset locations (path) and metadata (wav file for audio, and bid-format MEG data for brain signal).
- $sensorInfo$: metadata describing sensor identities and count C.
- $initialized$: $\mathbb{B}$, indicates whether the module has loaded the MEG dataset structure.

7.4.2 Environment Variables

- MEG recordings and metadata stored in the external file system.
- Dataset directory paths provided by the Input Parameters Module.

7.4.3 Assumptions

- The MEG dataset is available at the configured location.
- Each pair of subject identifier and utterance identifier uniquely identifies one MEG trial.
- The stimulus linkage between MEG trials and utterance identifiers is valid and consistent.

7.4.4 Access Routine Semantics

initMEGData():

- transition: $subjectIndex$, $trialIndex$, $sensorInfo$, and $initialized$ are constructed from the supplied MEG dataset configuration, and $initialized :=$ true.
- exception: raised if the MEG dataset is unavailable, unsupported, or cannot be indexed.

getSubjectList():

- output: returns the sequence of subject identifiers available in $\mathcal{D}_{\text{MEG}}$.
- exception: raised if the module has not been initialized.

getMEGResponse(subject id, utterance id):

- output: Read the preprocessed MEG trial from the external file system, and returns the response matrix $\mathbf{Y} \in \mathbb{R}^{N \times C}$, where N is the number of time samples in the trial and C is the number of MEG sensors defined by $sensorInfo$.

- exception: raised if the subject identifier or utterance identifier is unknown, or if the corresponding recording cannot be read.

getSensorCount():

- output: looks up dataset in $sensorInfo$ and returns the sensor count C stored in the corresponding sensor descriptor.
- exception: raised if the module has not been initialized.

getMEGSamplingRate():

- output: returns the MEG sampling frequency f_s.
- exception: raised if the module has not been initialized.

getTrialMetadata(subject id, utterance id):

- output: returns metadata associated with the given trial, such as timing information, recording identifiers, and preprocessing descriptors.
- exception: raised if the subject identifier or utterance identifier is unknown.

getStimulusLink(subject id, utterance id):

- output: returns the stimulus identifier linked to the requested MEG trial.
- exception: raised if the subject identifier or utterance identifier is unknown.

8 MIS of Input Parameters Module

8.1 Module

Input Parameters Module

8.2 Uses

None

8.3 Syntax

8.3.1 Exported Constants

- *SupportedDatasets*: the set of supported dataset identifiers.
- *SupportedTasks*: the set of supported task identifiers.
- *SupportedLabelTypes*: the set of supported label-type identifiers.
- *SupportedPolicies*: the set of supported policy identifiers.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
getPreset	preset name: string, seed: int	TrainConfig	unknown preset
validateConfig	cfg: TrainConfig	–	invalid numeric range, invalid categorical value, invalid field combination, file not found
ensureDirs	cfg: TrainConfig	–	directory creation failure
runId	cfg: TrainConfig	string	–
getRunDir	cfg: TrainConfig	Path	–
getCheckpointDir	cfg: TrainConfig	Path	–
getLogDir	cfg: TrainConfig	Path	–

8.4 Semantics

8.4.1 State Variables

- $presetTable$: a mapping from preset names to preset-construction routines.
- cfg: the current training configuration object.

8.4.2 Environment Variables

- Optional local path constants imported from the local path-definition file.

8.4.3 Assumptions

- A requested preset name is intended to identify one preset-construction routine in $presetTable$.
- When path checking is enabled, the required dataset paths and manifest paths exist in the external file system.

8.4.4 Access Routine Semantics

getPreset(preset name, seed):

- output: looks up the requested preset name in $presetTable$, invokes the corresponding preset-construction routine with the supplied seed, and returns a `TrainConfig`.
- exception: raised if the preset name is unknown.

validateConfig(cfg):

- output: checks that the configuration satisfies the required numeric, categorical, and combination constraints, and optionally checks dataset paths.
- exception: raised if any configuration field violates a range constraint, categorical constraint, combination constraint, or path-validity requirement.

ensureDirs(cfg):

- output: creates the run directory, checkpoint directory, and log directory associated with the supplied configuration.
- exception: raised if any required directory cannot be created.

runId(cfg):

- output: returns the canonical run identifier derived from dataset, task, label type, policy, and seed.

getRunDir(cfg):

- output: returns the run directory path associated with the supplied configuration.

getCheckpointDir(cfg):

- output: returns the checkpoint directory path associated with the supplied configuration.

getLogDir(cfg):

- output: returns the log directory path associated with the supplied configuration.

8.4.5 Local Functions

- **_coerce_enum**: converts user-supplied values into the required enum type.
- **_validate_paths**: checks dataset and manifest paths when path checking is enabled.
- **_apply_task_defaults**: fills task-specific default values in the configuration object.

9 MIS of Model Training Module

9.1 Module

Model Training Module

9.2 Uses

Input Parameters Module, Audio Data Module

9.3 Syntax

9.3.1 Exported Constants

- $\mathcal{C}_{\text{Train}}$: the abstract type of a training configuration.
- $\mathcal{R}_{\text{Eval}}$: the abstract type of an evaluation result record.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
getTrainer	cfg: $\mathcal{C}_{\text{Train}}$	training routine	unsupported task or policy
runOnce	cfg: $\mathcal{C}_{\text{Train}}$, device: string	result summary file and checkpoint files	invalid configuration, loader failure, training failure, file error
buildLoaders	cfg: $\mathcal{C}_{\text{Train}}$, vocab, tailFrames: int	train loader, validation loader, test loader	dataset unavailable, malformed manifest, loader construction failure
runEval	model, loader, device: string, srvTable, cfg: $\mathcal{C}_{\text{Train}}$, tailFrames: int	$\mathcal{R}_{\text{Eval}}$	evaluation failure, malformed batch
saveCheckpoint	path: Path, model, optimizer, epoch: int, bestVal: real	–	file error

9.4 Semantics

9.4.1 State Variables

- cfg: the active training configuration.
- $bestVal$: the best validation loss observed so far.
- $currentEpoch$: the current training epoch.

9.4.2 Environment Variables

- Checkpoint files, logs, and result files stored in the external file system.
- The hardware device selected for training or evaluation (for example, CPU or CUDA device).

9.4.3 Assumptions

- The supplied configuration has already been validated by the Input Parameters Module.
- The requested training dataset and manifests are available through the Audio Data Module.
- The selected model architecture is compatible with the batch feature dimension produced by the Audio Data Module.

9.4.4 Access Routine Semantics

getTrainer(cfg):

- output: returns the training routine associated with the task and policy specified in cfg.
- exception: raised if no supported training routine exists for the requested configuration.

runOnce(cfg, device):

- transition: creates output directories, initializes logging, constructs the vocabulary and data loaders, initializes the model and optimizer, runs the training loop, evaluates the best checkpoint on the test split, and writes checkpoint and result files.
- output: produces checkpoint files, logs, and a result summary for the completed training run.
- exception: raised if configuration validation, loader construction, training, evaluation, or output serialization fails.

buildLoaders(cfg, vocab, tailFrames):

- output: returns the train, validation, and test data loaders implied by the current configuration and vocabulary.
- exception: raised if the required manifests or dataset subsets are unavailable or malformed.

runEval(model, loader, device, srvTable, cfg, tailFrames):

- output: evaluates the model on the supplied loader and returns an evaluation-result record containing loss and summary metrics.
- exception: raised if evaluation encounters malformed inputs or invalid tensor shapes.

saveCheckpoint(path, model, optimizer, epoch, bestVal):

- output: writes a checkpoint file containing model state, optimizer state, epoch number, and the best validation value observed so far.
- exception: raised if the checkpoint file cannot be written.

9.4.5 Local Functions

- **srvLossAndCosFrame**: computes the frame-level SRV loss and cosine summary used during training and evaluation.
- **srvLossAndCosWord**: computes the word-level semantic SRV loss and cosine summary used during training and evaluation.

10 MIS of Frozen Model Module

10.1 Module

Frozen Model Module

10.2 Uses

Input Parameters Module, Model Training Module

10.3 Syntax

10.3.1 Exported Constants

- $\mathcal{M}_{\text{Frozen}}$: the abstract type of a frozen neural model.

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
loadCheckpoint Model	checkpoint path: Path, device: string	$\mathcal{M}_{\text{Frozen}}$	checkpoint unavailable, incompatible checkpoint, file error
freezeModel	model	$\mathcal{M}_{\text{Frozen}}$	invalid model instance
getLayerCount	frozen model: $\mathcal{M}_{\text{Frozen}}$	L	invalid model instance
isFrozen	frozen model: $\mathcal{M}_{\text{Frozen}}$	$\mathbb{B}$	invalid model instance

10.4 Semantics

10.4.1 State Variables

- *loadedCheckpoint*: the checkpoint currently loaded by the module.
- $frozenModel$: the non-trainable model instance derived from the loaded checkpoint.

10.4.2 Environment Variables

- Stored checkpoint files in the external file system.

10.4.3 Assumptions

- A valid trained checkpoint exists at the supplied checkpoint path.
- The checkpoint contains sufficient information to reconstruct the model architecture and load its parameters.

10.4.4 Access Routine Semantics

loadCheckpointModel(checkpoint path, device):

- transition: reads the checkpoint file, reconstructs the corresponding model instance, loads the stored parameters, converts the model to evaluation mode, and stores it as $frozenModel$.
- output: returns the frozen model instance.
- exception: raised if the checkpoint is unavailable, incompatible, or cannot be loaded.

freezeModel(model):

- output: disables parameter updates for the supplied model and returns the frozen model instance.
- exception: raised if the supplied model is invalid.

getLayerCount(frozen model):

- output: returns the number of accessible internal layers exposed by the frozen model.
- exception: raised if the supplied model is invalid.

isFrozen(frozen model):

- output: returns `true` if the supplied model is in non-trainable frozen form, and `false` otherwise.
- exception: raised if the supplied model is invalid.

10.4.5 Local Functions

- **inferModelFromCheckpoint**: reconstructs the model architecture associated with a stored checkpoint.
- **loadStateDict**: loads stored parameters into the reconstructed model.

11 MIS of Representation Extraction Module

11.1 Module

Representation Extraction Module

11.2 Uses

Input Parameters Module, Audio Data Module, Frozen Model Module, MEG Data Module

11.3 Syntax

11.3.1 Exported Constants

- $\mathcal{H}$: the abstract type of a hidden-state sequence.
- $\mathcal{R}$: the abstract type of a time-by-feature predictor matrix.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
forwardWithHidden	frozen model, input tensor, input lengths	hidden-state sequence $\in \mathcal{H}$	model execution failed, invalid input shape
computePredictors FromHidden	hidden-state sequence $\in \mathcal{H}$, transform specification	predictor array(s)	invalid hidden-state shape, transform failed
alignToAnalysisGrid	predictor array(s), analysis time grid	predictor matrix $\in \mathcal{R}$	time-grid mismatch, alignment failed
serializePredictors	predictor matrix $\in \mathcal{R}$, output path specification	saved predictor file(s)	file error, serialization failed

11.4 Semantics

11.4.1 State Variables

- *extractionSpec*: the extraction and transformation parameters supplied by the Input Parameters Module.
- *timeGrid*: the analysis time grid used to align predictor outputs.

11.4.2 Environment Variables

- Input feature files and serialized predictor files stored in the external file system.

11.4.3 Assumptions

- The required frozen model has already been loaded.
- The requested input sequence has a feature dimension compatible with the frozen model.
- The analysis time grid is available and compatible with the predictor length.

11.4.4 Access Routine Semantics

forwardWithHidden(frozen model, input tensor, input lengths):

- output: applies the frozen model to the supplied input sequence and returns the hidden-state sequence exposed by the model.
- exception: raised if the model is unavailable or the input tensor has an invalid shape.

computePredictorsFromHidden(hidden-state sequence, transform specification):

- output: applies the requested transformation to the hidden-state sequence and returns one or more predictor arrays derived from the hidden representations.
- exception: raised if the hidden-state sequence is invalid or the requested transformation cannot be applied.

alignToAnalysisGrid(predictor array(s), analysis time grid):

- output: converts the predictor array(s) into predictor matrices aligned to the requested analysis time grid.
- exception: raised if the predictor length and the analysis time grid are incompatible.

serializePredictors(predictor matrix, output path specification):

- output: writes the aligned predictor matrix to the required serialized output format.
- exception: raised if serialization fails or the output files cannot be written.

11.4.5 Local Functions

- **computeMagnitudeChange**: computes magnitude-based and change-based predictor arrays from hidden states.
- **wrapToSerializedPredictor**: converts predictor arrays into the serialized predictor format expected by downstream analysis.

12 MIS of Statistical Analysis Module

12.1 Module

Statistical Analysis Module

12.2 Uses

Input Parameters Module, MEG Data Module, Representation Extraction Module

12.3 Syntax

12.3.1 Exported Constants

- $\mathcal{T}_{\text{TRF}}$: the abstract type of a TRF analysis result.
- $\mathcal{S}_{\text{Summary}}$: the abstract type of a statistical summary table.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
runTRFJob	analysis configuration	$\mathcal{T}_{\text{TRF}}$	invalid configuration, predictor unavailable, fitting failed
loadPredictorSafe	predictor identifier, predictor source	predictor object	predictor unavailable, nonfinite values, malformed predictor
summarizeTRFResults	TRF result $\in \mathcal{T}_{\text{TRF}}$	$\mathcal{S}_{\text{Summary}}$	result unavailable, malformed result
computeImprovementTable	baseline result, target result	improvement summary table	shape mismatch, comparison failed
selectBestLayer	summary table, selection criterion	layer index	invalid criterion, empty summary

12.4 Semantics

12.4.1 State Variables

- *analysisSpec*: the statistical-analysis configuration supplied by the Input Parameters Module.
- *resultCache*: a mapping from analysis identifiers to stored TRF result objects.

12.4.2 Environment Variables

- Cached TRF result files stored in the external file system.

12.4.3 Assumptions

- The required MEG data and aligned predictors are already available.
- The analysis configuration defines a valid lag window, sampling rate, and cross-validation setting.
- The underlying Eelbrain/TRF pipeline is available to execute the requested analysis.

12.4.4 Access Routine Semantics

runTRFJob(analysis configuration):

- transition: executes the configured TRF analysis, stores the resulting TRF object in *resultCache*, and records any generated cache files.
- output: returns the TRF result object for the requested analysis job.
- exception: raised if the supplied analysis configuration is invalid or if TRF execution fails.

loadPredictorSafe(predictor identifier, predictor source):

- output: loads the requested predictor, checks for non-finite values and degenerate columns, and returns a predictor object suitable for downstream TRF analysis.
- exception: raised if the predictor is unavailable, malformed, or unusable.

summarizeTRFResults(TRF result):

- output: extracts the summary quantities required for downstream reporting, comparison, and plotting.
- exception: raised if the supplied TRF result is malformed or incomplete.

computeImprovementTable(baseline result, target result):

- output: returns the baseline-referenced comparison table between the supplied analysis results.
- exception: raised if the supplied results are incompatible.

selectBestLayer(summary table, selection criterion):

- output: returns the layer index that optimizes the supplied selection criterion.
- exception: raised if the summary table is empty or the criterion is invalid.

12.4.5 Local Functions

- **labelEvents**: matches event markers to stimulus structure and constructs the event labels used for epoching.
- **sanitizePredictor**: replaces non-finite predictor values and detects flat columns before analysis.

13 MIS of Plotting Module

13.1 Module

Plotting Module

13.2 Uses

Input Parameters Module, Statistical Analysis Module

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
plotCurves	cosine curves, probability curves, L, env sampling rate, output path, title	saved figure path	invalid plotting input, plotting failed, file error
buildPlotSpec	analysis summary, plotting options	plot specification	invalid summary, invalid plotting options
saveFigure	plot specification, output path	saved figure path	plotting failed, file error

13.4 Semantics

13.4.1 State Variables

- $figureRegistry$: a mapping from plot identifiers to saved figure paths.

13.4.2 Environment Variables

- Figure files written to the external file system.

13.4.3 Assumptions

- The statistical summaries or curve arrays required for plotting are already available.
- The output directory for the requested figure is writable.

13.4.4 Access Routine Semantics

plotCurves(cosine curves, probability curves, L, env sampling rate, output path, title):

- transition: renders the supplied curve arrays into a figure and stores the saved figure path in $figureRegistry$.
- output: returns the path of the generated figure file.
- exception: raised if the supplied curve arrays are malformed or the figure cannot be written.

buildPlotSpec(analysis summary, plotting options):

- output: returns the internal plotting specification derived from the supplied analysis summary and plotting options.
- exception: raised if the summary or plotting options are invalid.

saveFigure(plot specification, output path):

- transition: renders the supplied plotting specification and writes the corresponding figure file to the requested output path.
- output: returns the saved figure path.
- exception: raised if rendering fails or the output path is invalid.

13.4.5 Local Functions

- **renderFigure**: converts the plotting specification into a concrete figure object.
- **registerFigure**: records the saved figure path in $figureRegistry$.

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.