NeuralSet: A High-Performing Python Package for Neuro-AI

Jean-Rémi King[1], **Corentin Bel**[1,2,*], **Linnea Evanson**[3,*], **Julien Gadonneix**[3,5,*], **Sophia Houhamdi**[1,4,*], **Jarod Lévy**[1,4,*], **Josephine Raugel**[1,2,*], **Andrea Santos Revilla**[3,5,*], **Mingfang (Lucy) Zhang**[2,3,*], **Julie Bonnaire**[6], **Charlotte Caucheteux**[1], **Alexandre Défossez**[1], **Théo Desbordes**[1], **Pablo Diego-Simón**[2], **Shubh Khanna**[1], **Juliette Millet**[1], **Pierre Orhan**[1], **Saarang Panchavati**[1], **Antoine Ratouchniak**[1], **Alexis Thual**[1], **Teon L. Brooks**[1,†], **Katelyn Begany**[1,†], **Yohann Benchetrit**[1,†], **Marlène Careil**[1,†], **Hubert Banville**[1,†], **Stéphane d'Ascoli**[1,†], **Simon Dahan**[1,†], **Jérémy Rapin**[1]

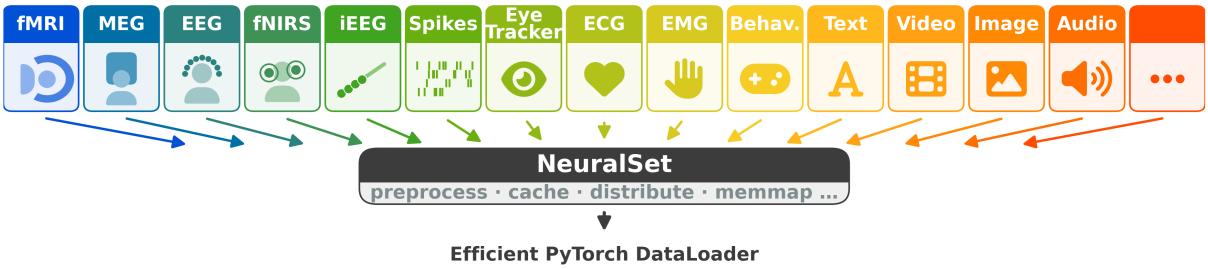
[1]FAIR, Meta, [2]École Normale Supérieure - PSL Université, Paris, [3]Foundation Adolphe de Rothschild Hospital, Paris, [4]MIND, Inria, [5]Université Paris Cité, [6]Neurospin, CEA, Gif sur Yvette
[*]These authors contributed equally., [†]These senior authors contributed equally.

Artificial intelligence (AI) is increasingly central to understanding how the brain processes information. However, the integration of neuroscience and modern AI is bottlenecked by a fragmented software ecosystem. Current tools are siloed by recording modality and optimized for small-scale, in-memory workflows, limiting the use of massive, naturalistic datasets. Here, we introduce NeuralSet, a Python framework that efficiently unifies the processing of diverse neural recordings (including fMRI, M/EEG, and spikes) and complex experimental stimuli (such as text, audio, and video). By decoupling experimental metadata from lazy, memory-efficient data extraction, NeuralSet harmonizes standard neuroscientific preprocessing pipelines with pretrained deep learning embeddings. This approach provides a single PyTorch-ready interface that scales seamlessly from local prototyping to high-performance cluster execution. By eliminating manual data wrangling and ensuring full computational provenance, NeuralSet establishes a scalable, unified infrastructure for the next generation of neuro-AI research.

Correspondence: jeanremi@meta.com, jrapin@meta.com

Meta

A



B

```python
from neuralset import Study, Segmenter
import neuralset.extractors as nsx

events = Study(name="MyStudy", path="my/path").build()  # -> self-contained pd.DataFrame

neuro = nsx.FmriExtractor(frequency=1, detrend=True, input_space="MNI152")
# neuro = nsx.MegExtractor(frequency=120, filter=(0.5, 40), scaler="robust")
# neuro = nsx.IeegExtractor(reference="bipolar", filter=(70, 150), apply_hilbert=True)
stim  = nsx.HuggingFaceText(model_name="gpt2", layers=(8, 9, 10, 11), contextualized=True)
# stim  = nsx.HuggingFaceImage(model_name="facebook/dinov2-base", token_aggregation="mean")
# stim  = nsx.HuggingFaceAudio(model_name="facebook/wav2vec2-large-xlsr-53")

seg = Segmenter(start=-0.5, duration=1.0, stride=10, extractors=dict(neuro=neuro, stim=stim))
dataset = seg.apply(events)  # -> PyTorch Dataset
```

Figure 1 **A.** NeuralSet unifies the processing of neural recordings and the AI embedding of experimental conditions into a single PyTorch DataLoader. **B.** Specifying the extraction of any types of data requires changing the configuration of a single "extractor".

1 Introduction

The rise of naturalistic Neuro-AI. Neuroscience is undergoing a paradigm shift, transitioning from highly controlled, small-scale experiments to massive, multi-modal, and naturalistic datasets (Hasson et al., 2004; Sonkusare et al., 2019). This evolution is essential for understanding the biological foundations of intelligence in real-world contexts and effectively fuels the emergence of new discipline: Neuro-AI (Richards et al., 2019; Yamins and DiCarlo, 2016). Yet, the application of deep learning onto neuroscientific data (Caucheteux and King, 2022; Défossez et al., 2023; Schrimpf et al., 2020) is severely bottlenecked by a fragmented software ecosystem, which leads most labs across the world, to re-implement similar home-made data processing workflows. As public datasets reach the terabyte scale[1] and experimental protocols increasingly incorporate complex stimuli like continuous speech and video (Allen et al., 2022; Gwilliams et al., 2023; Nastase et al., 2021), the infrastructure required to process these data has failed to keep pace.

A rich but fragmented software landscape. Over the past two decades, the neuroscience community has built a large set of open-source tools that have standardized analysis practices and enabled thousands of studies. Established softwares such as MNE-Python (Gramfort et al., 2013), EEGLAB (Delorme and Makeig, 2004), FieldTrip (Oostenveld et al., 2011), and Brainstorm (Tadel et al., 2011) for electrophysiology, or Nilearn (Abraham et al., 2014) and fMRIPrep (Esteban et al., 2019) for neuroimaging, are the gold standard for signal processing and statistical inference. However, these tools were originally designed for a pre-deep-learning era: they typically rely on eager loading – assuming datasets can fit entirely into RAM – and lack native abstractions to temporally align neural time series with high-dimensional, pretrained embeddings from modern AI frameworks (e.g., HuggingFace transformers (Wolf et al., 2020)). Consequently, researchers must build ad-hoc pipelines that require manual data wrangling, manual caching, and complex backend configurations.

Structure–data decoupling. To bridge this gap, we introduce NeuralSet, a Python package designed to efficiently unify the processing of diverse neural recordings and their associated experimental conditions. NeuralSet is built upon the principle of structure-data decoupling: it represents the logical structure of any experiment as lightweight, event-driven metadata, separating it entirely from the memory- and compute-intensive extraction of the underlying signals. This architecture enables lazy, on-demand data processing, allowing researchers to filter, recombine, and iterate on massive datasets without loading a single byte of raw data into memory until it is required by a PyTorch dataloader (Paszke et al., 2019).

A unified, scalable interface. By providing a single, backend-agnostic interface, NeuralSet harmonizes the validated preprocessing stacks of domain-specific libraries with the embedding capabilities of modern AI models. It natively supports a comprehensive array of neural modalities: e.g. fMRI, EEG, MEG, iEEG, fNIRS, EMG, and spike trains, alongside the embedding of naturalistic text, audio, and video stimuli. Critically, its caching and hardware-agnostic execution relies on exca (Rapin and King, 2024), which allows researchers to scale seamlessly from local prototyping on a laptop to distributed execution on high-performance computing clusters. Ultimately, NeuralSet is designed to eliminate the friction of multi-task and multi-device data orchestration, providing the scalable and reproducible infrastructure necessary for the next generation of Neuro-AI research.

2 Framework

NeuralSet enforces a separation between the lightweight description of an experiment and the heavy extraction of data. The framework is organized around five core abstractions – `Events`, `Extractors`, `Segments` and `Tensor Data` – which together form a pipeline from raw neural and/or stimulus files to PyTorch-ready datasets.

2.1 Events: describing what happens and when.

The central insight of NeuralSet is that the logical structure of an experiment can be represented independently of the underlying data. Every piece of information – an fMRI run, a word spoken by a narrator, a stimulus

[1] https://openneuro.org/

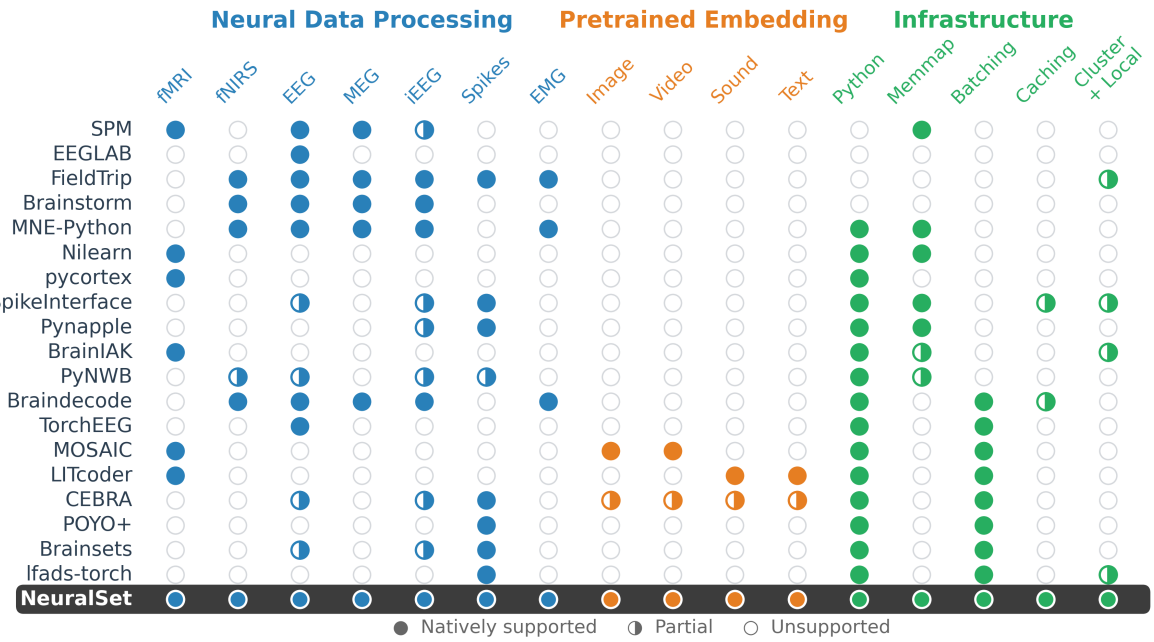


Figure 2 Comparison of neuroscience software packages across neural devices, experimental tasks, and infrastructure features. Filled circles indicate full support, half-filled circles indicate partial or compatible support, and empty circles indicate no support. NeuralSet (bottom row, highlighted) is the only package supporting all categories.

image – is modelled as an *event*: a lightweight object (i.e. a Python dictionary) defined by a `type`, a `start` time, a `duration`, and a `timeline` (a unique identifier for a continuous recording session). For instance, a one-hour fMRI session in which a participant watches a movie is described by a handful of concurrent events on the same timeline: the fMRI acquisition, the video stimulus, and potentially hundreds of word-onset annotations derived from the soundtrack.

A `Study` object assembles the events of an entire dataset into a single pandas DataFrame. This approach is consistent with, but not restricted to BIDS-compliant datasets (Gorgolewski et al., 2016). Because the DataFrame contains only light metadata – not the signals themselves – researchers can explore, filter, and recombine experiments using standard pandas (McKinney, 2010) operations (e.g. selecting all timelines of a given subject, or all word events in a specific language) without loading data into memory.

Composable `EventsTransform` operations can then be chained to enrich, filter, or reorganize events prior to data extraction. Typical transforms include annotating words with their sentence context, assigning cross-validation splits, or chunking long audio and video events into shorter segments.

2.2 Extractors: turning events into tensors.

`Extractors` bridge the gap between the metadata layer and the numerical arrays required by modern machine-learning models. Given the events that fall within a temporal window, an extractor reads, preprocesses, and returns a dense tensor.

For neural recordings, NeuralSet wraps the established preprocessing stacks of domain-specific libraries. For example, an fMRI extractor leverages Nilearn (Abraham et al., 2014) for signal cleaning, spatial smoothing, and surface or atlas-based projection, while an EEG or MEG extractor delegates to MNE-Python (Gramfort et al., 2013) for filtering, re-referencing, and resampling. The same interface covers iEEG, fNIRS, EMG, and spike recordings, so that switching modalities requires only changing a configuration parameter, not rewriting a pipeline.

Study Define a data source

```python
import neuralset as ns
study = ns.Study(name="FakeMegStudy2025", path=ns.CACHE_FOLDER)
events = study.build()  # -> pd.DataFrame
```

Events Every row is a self contained event

```python
import pandas as pd
events = pd.DataFrame([
    dict(type="Meg",   start=0.0, duration=600.0, timeline="sub-01_run-01", filepath="meg.fif"),
    dict(type="Word",  start=1.2, duration=0.5,   timeline="sub-01_run-01", text="The"),
    dict(type="Word",  start=1.7, duration=0.6,   timeline="sub-01_run-01", text="cat"),
    dict(type="Image", start=5.0, duration=2.0,   timeline="sub-01_run-01", filepath="img_01.png"),
])
```

Transforms Sequentially filter, annotate, or chunk events

```python
from neuralset.events import transforms
transform = transforms.AddSentenceToWords()
events = transform(events)
transform = transforms.QueryEvents(query='type=="Word"')
events = transform(events)
```

Extractors Tensor extraction from events — preprocessing / embedding

```python
meg  = ns.extractors.MegExtractor(frequency=100.0)
tensor = meg(events, start=0.0, duration=1.0)   # 1 × C × T
text = ns.extractors.HuggingFaceText(model_name="gpt2", frequency=0.)
tensor = text(events, start=0.0, duration=1.0)  # 1 × D
```

Segmenter Slice events into segments, wire extractors

```python
segmenter = ns.dataloader.Segmenter(
    start=-0.1, duration=0.5, trigger_query='type=="Word"',
    extractors=dict(meg=meg, text=text),
)
dataset = segmenter.apply(events)  # -> SegmentDataset
print(dataset.segments[0])  # events used by extractors and overlapping with [1.1, 1.6]
```

Batch Standard PyTorch batching via Segments

```python
from torch.utils.data import DataLoader
dataset.prepare()  # precompute + cache on Slurm / local
loader = DataLoader(dataset, batch_size=32, collate_fn=dataset.collate_fn)
for batch in loader:  # first time heavy data are actually loaded
    x = batch.data["meg"]   # B × C × T
    y = batch.data["text"]  # B × D
```

Figure 3 The NeuralSet pipeline from user perspective. Each step is shown alongside its Python code snippet, from defining a data source (`Study`) to iterating batched tensors in a standard PyTorch `DataLoader`. The entire pipeline is lazy: only the final iteration triggers data loading and feature extraction.

For naturalistic stimuli, NeuralSet provides native integration with the HuggingFace ecosystem. A single `HuggingFaceImage` extractor can embed every stimulus frame through DINOv2 (Oquab et al., 2023), CLIP (Radford et al., 2021), or any compatible vision model; analogous extractors exist for audio (Wav2Vec (Baevski et al., 2020), Whisper (Radford et al., 2023)), text (GPT-2 (Radford et al., 2019), LLaMA (Touvron et al., 2023)), and video (VideoMAE (Tong et al., 2022)). Critically, NeuralSet can expand a static embedding (e.g. a single vector per image) into a time series at an arbitrary frequency, so that stimulus representations are always temporally aligned with neural recordings.

Extractors follow a three-phase execution model: `configure` (parameter validation at construction), `prepare` (pre-compute and cache heavy outputs for all events), and `extract` (lazy retrieval from cache during model training). This design ensures that expensive computations — such as running a large language model over every word in a corpus — are performed once and reused across experiments.

A key abstraction is the distinction between the *intrinsic nature* of the data and the way an extractor *uses* it along the time axis. Intrinsically *static* data (e.g. a word embedding) yields a single D-dimensional vector per event, whereas intrinsically *dynamic* data (e.g. a mel spectrogram) yields a $D \times T$ matrix that varies continuously over time. Crucially, NeuralSet allows any extractor to produce its output either statically or dynamically, regardless of the data's intrinsic nature (Fig. 4).

2.3 Segments: from events to a PyTorch dataset.

A `Segment` is a contiguous temporal window over a set of events, representing a single training example. NeuralSet provides a `Segmenter` that slices the events DataFrame into segments — either on a regular grid (sliding window) or anchored to specific *trigger* events such as image or word onsets. Each segment retains a reference to its trigger, so that extractors can distinguish, for example, the particular word that anchored a two-second window from the other words that happen to fall within it. Because slicing operates entirely in the metadata domain, millions of segments can be prepared without touching the raw signal files.

Before any data is loaded, NeuralSet validates that the required events are present and that all extractors have been prepared, catching configuration errors early rather than hours into a processing run. The resulting `SegmentDataset` is a standard PyTorch `Dataset`: it can be wrapped in a `DataLoader`, passed to PyTorch Lightning, or consumed by any framework that expects the PyTorch data interface.

2.4 Tensor Data: the []nal product.

The output of an `Extractor` for a single segment, containing a dictionary of tensors keyed by extractor name, as well as the corresponding segments. SegmentDataset only contains the list of lightweight segments and the configured extractors. SegmentDataset.prepare allows to precompute the data for all segments in one go.

2.5 Backend

NeuralSet is built upon the `exca` package – designed to seamlessly orchestrate the configuration, validation, and execution of hierarchical pipelines.

Proactive Configuration Validation. To eliminate "late-stage" failures – errors that occur hours into a processing run, NeuralSet utilizes Pydantic to enforce strict schema validation at initialization. For example, if a user specifies a negative filter frequency, or provides an invalid path for a BIDS directory, the framework raises a validation error immediately, i.e. before job submission. This proactive check ensures that all parameters, recording modalities, and hardware constraints are logically consistent before any I/O or resource-intensive computation begins.

Deterministic and Incremental Caching. NeuralSet implements a hierarchical hash-based caching system, where the state of each pipeline stage is tethered to its specific non-default parameters and its relevant preceding dependencies.

A

```python
# Events
events = pd.DataFrame([
  dict(type="Word",  start=1.0,  duration=.3,   timeline="x", text="This"),
  dict(type="Word",  start=1.35, duration=.2,   timeline="x", text="is"),
  dict(type="Word",  start=1.6,  duration=.15,  timeline="x", text="a"),
  dict(type="Word",  start=1.8,  duration=.5,   timeline="x", text="sentence"),
  dict(type="Audio", start=.72,  duration=.85,  timeline="x", filepath="this_is.wav"),
  dict(type="Audio", start=1.57, duration=.93,  timeline="x", filepath="a_sentence.wav"),
])

# Static Extractor
static  = SpacyEmbedding(aggregation="trigger")     # D × 1
dynamic = SpacyEmbedding(frequency=100)             # D × T
# Dynamic Extractor
dynamic = MelSpectrum(frequency=100, n_mels=40)     # D × T
static  = ToStatic(dynamic, aggregation="trigger")  # D × 1
```

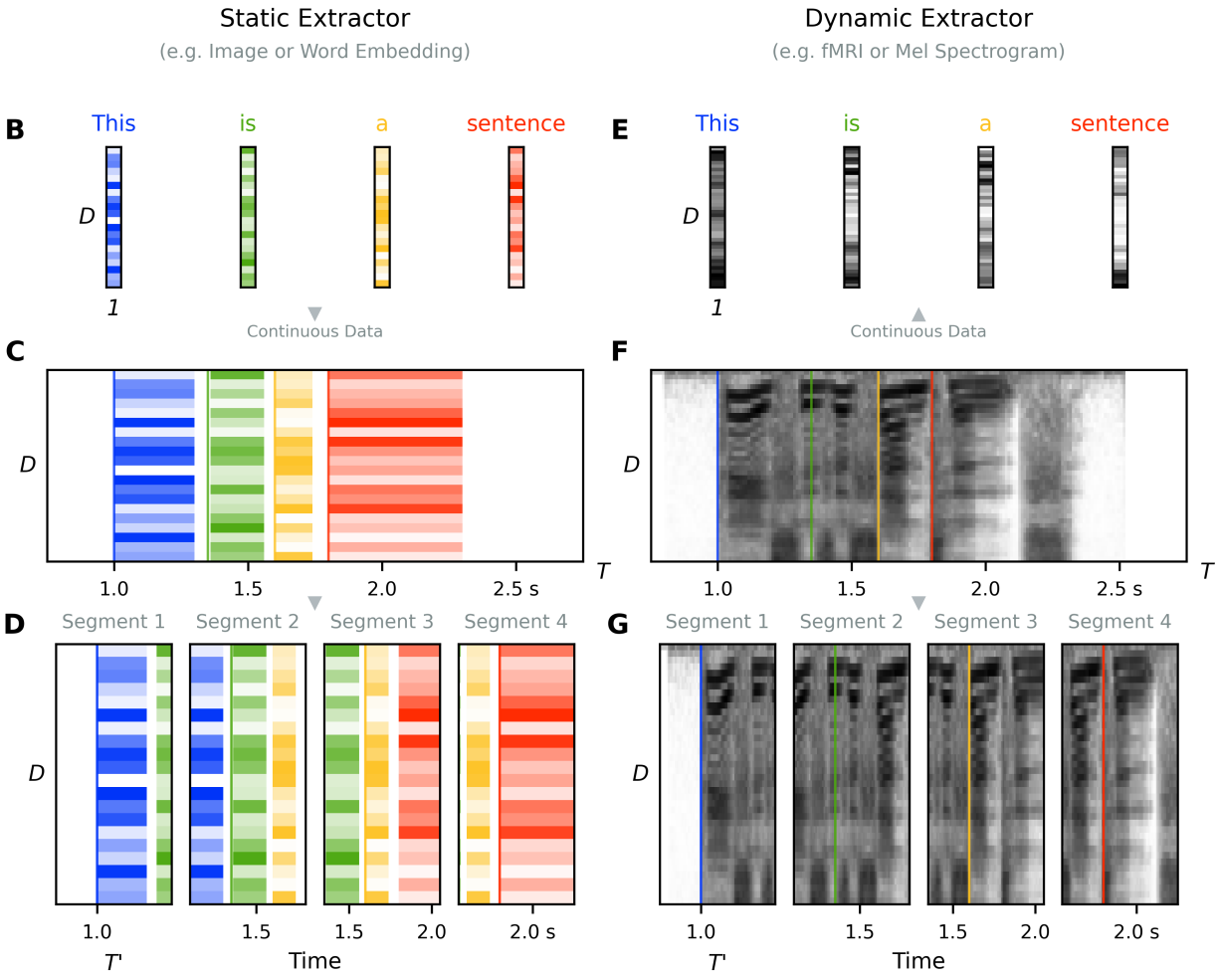


Figure 4 Static and dynamic extractors applied to word and audio events. **A.** Python snippet defining the events DataFrame and two extractors: a static word embedding and a dynamic mel spectrogram. **B–D.** Static extractor: each word is mapped to a single D-dimensional vector (B), placed on a shared timeline (C), and segmented around each trigger onset (D). **E–G.** Dynamic extractor: each word onset is sliced from the mel spectrogram into a static D-dimensional snapshot (E), the full spectrogram is shown on the same timeline (F), and trigger-aligned segments are extracted (G). Vertical dashed lines mark word onsets; the "to dynamic" and "to static" arrows illustrate how NeuralSet converts between representations.

- **Parameter-Aware Storage:** The cache key for a given step (e.g., MEG filtering) is a deterministic function of both its local configuration and the upstream data state. Changing a single parameter, such as a smoothing kernel width, invalidates only the downstream cache, leaving independent branches untouched.
- **Full Provenance:** By persisting intermediate results alongside their metadata, NeuralSet ensures that any processed tensor can be traced back to the exact version of the raw data and the specific preprocessing chain used to generate it, facilitating perfect reproducibility across different research environments.

Hardware Abstraction and Portability. A core strength of the framework is its ability to decouple the analysis logic from the underlying compute infrastructure. NeuralSet provides a backend-agnostic interface that allows the same script to be executed across varying environments simply by updating a configuration flag. When configured for a cluster, the framework automatically handles job submission, environment synchronization (e.g., Conda/Singularity), and file path translation. Because the cache is shared, a researcher can prototype an extractor on a single local subject and then dispatch the remaining 100 subjects to a SLURM-based HPC cluster without writing any infrastructure-specific code.

Figure 2 compares NeuralSet with existing packages across these dimensions.

A. YAML Configuration

```yaml
study:
  name: Gwilliams2022
  path: /data
transforms:
  - name: AddSentences
segmenter:
  trigger_query: "type == 'Word'"
  extractors:
    meg:
      frequency: 120
      []lter: [0.5, 30]
    text:
      model_name: gpt2
      frequency: 0.
```

B. Hierarchical Configuration

```python
import pydantic, yaml
from neuralset import StudyLoader
from neuralset import EventTransform
from neuralset import Segmenter
from neuralset.events.transforms import AddSentences

class DatasetBuilder(pydantic.BaseModel):
    study: StudyLoader
    transforms: list[EventTransform]
    segmenter: Segmenter

    def run(self) -> SegmentDataset:
        events = self.study.build()
        for t in self.transforms:
            events = t(events)
        return self.segmenter.apply(events)

# Load from YAML config (A)
cfg = yaml.safe_load(open("config.yaml"))
exp = DatasetBuilder(**cfg)
dataset = exp.run()
```

C. Caching & SLURM Dispatch

```python
import pydantic, exca

class Preprocessor(pydantic.BaseModel):
    smooth: float = 6.0
    infra: exca.TaskInfra = exca.TaskInfra()

    @infra.apply
    def run(self) -> np.ndarray:
        ...  # expensive computation
        return out

# Caching: first call saves, second loads
p = Preprocessor(smooth=6.0,
    infra=dict(folder="/cache"))
p.run()  # computes and caches
p.run()  # loaded from cache

# SLURM: same code, one config change
p = Preprocessor(smooth=6.0, infra=dict(
    folder="/cache", cluster="slurm"))
p.run()  # dispatched if not computed
```

Figure 5 Backend infrastructure. **A.** Example YAML configuration specifying the study, transforms, and extractors for a complete pipeline. **B.** Hierarchical configuration: every pipeline component is a Pydantic `BaseModel`, validated at init and loadable from YAML (A). **C.** Caching and SLURM dispatch: decorated methods are deterministically cached on first call; switching `cluster` to `"slurm"` dispatches jobs to HPC with no code change.

3 Discussion

Summary. The quest to understand the biological foundations of intelligence necessitates a fundamental shift from fragmented, task-specific analyses toward unified, large-scale modeling of neural dynamics (Richards et al., 2019; Yamins and DiCarlo, 2016). We introduced NeuralSet, a Python framework that bridges the gap

between raw, multi-modal neuroscientific recordings and modern deep learning architectures. By decoupling the logical structure of an experiment from the memory-intensive extraction of its underlying signals, NeuralSet provides a scalable, backend-agnostic infrastructure that eliminates the manual data wrangling traditionally required in Neuro-AI.

Orchestration, not reinvention. Unlike existing software suites that are optimized for modality-specific, eager-loading workflows (Gramfort et al., 2013; Abraham et al., 2014), NeuralSet acts as a specialized orchestrator. It does not reinvent validated signal processing algorithms; rather, it harmonizes them with the high-dimensional embeddings of contemporary AI models (Wolf et al., 2020). This intentional specialization ensures that researchers can leverage decades of peer-reviewed preprocessing techniques while seamlessly transitioning to PyTorch-ready datasets (Paszke et al., 2019). Furthermore, its deterministic caching and hardware-agnostic execution democratize access to high-performance computing, allowing complex pipelines to scale from local laptops to SLURM-based clusters without code modification.

Complementing the ecosystem. Critically, NeuralSet is not intended to supplant the established tools upon which modern neuroscience rests. Packages such as MNE-Python (Gramfort et al., 2013), Nilearn (Abraham et al., 2014), EEGLAB (Delorme and Makeig, 2004), FieldTrip (Oostenveld et al., 2011), Brainstorm (Tadel et al., 2011), and fMRIPrep (Esteban et al., 2019) embody decades of validated, peer-reviewed signal processing; NeuralSet neither reimplements nor shadows their algorithms. Instead, it occupies a complementary niche—the orchestration layer that sits between these preprocessing stacks and downstream deep-learning frameworks such as PyTorch or Braindecode (Schirrmeister et al., 2017). Concretely, every signal-processing step is still executed by the original library: NeuralSet only structures, caches, and batches its output. The modular `Extractor` abstraction makes this integration bidirectional: maintainers of any existing package can expose their pipeline as a NeuralSet backend through a thin adapter, thereby gaining access to lazy loading, deterministic caching, and cluster-level execution without modifying their own codebase. Likewise, the comparison presented in Figure 2 should be read not as a ranking but as a map of the ecosystem's current coverage: the gaps reflect the natural specialization of each tool, and NeuralSet's breadth stems precisely from its ability to delegate to these specialists. We therefore view this project as community infrastructure and actively invite package maintainers and domain experts to contribute extractors, propose new modalities, and shape the framework's roadmap.

Limitations. Despite its advantages, NeuralSet has limitations. Its reliance on underlying libraries (e.g., MNE-Python (Gramfort et al., 2013), Nilearn (Abraham et al., 2014)) means that it inherits their respective dependencies and performance bottlenecks during the initial extraction phase. Additionally, while the lazy-loading architecture drastically reduces RAM usage during model training, the initial preparation of heavily augmented or high-frequency datasets can still incur significant storage overhead due to the caching of intermediate tensors. Future development will focus on integrating streaming data formats and expanding native support for real-time, closed-loop experimental designs.

Beyond time series. In its current form, NeuralSet organizes every recording and stimulus along shared timelines, making it a natural fit for time-series data such as M/EEG, fMRI runs, and continuous audio or video. However, many neuroscientific workflows involve interactions between elements that do not share a common timeline. For example, MEG source reconstruction typically requires a T1-weighted anatomical MRI acquired in a separate session, and some structural scans are spatially large volumes that have no meaningful temporal dimension. Such cross-session and cross-modality dependencies are not yet first-class citizens in the framework. We anticipate that NeuralSet can scale to these use cases by leveraging relational database semantics—for instance through the lazy-frame and join capabilities of Polars (Vink, 2024)—so that the events dataframe need not be strictly bound to a single set of timelines but can instead reference auxiliary data through relational links.

Toward cross-modal foundation models. Ultimately, the main bottleneck in Neuro-AI is no longer a lack of data—with massive datasets increasingly available through standardized formats like BIDS (Gorgolewski et al., 2016)—but the lack of an integrated architecture to process it efficiently. NeuralSet represents a

departure from the era of "frozen" datasets and bespoke scripts. By unifying disparate domains—from single-unit electrophysiology to whole-brain neuroimaging—under a single, extensible framework, we provide the groundwork for neural foundation models that are unconstrained by the recording device or the experimental task. This infrastructure invites the global computational community to join neuroscientists in the pursuit of modeling the biological bases of intelligence.

Acknowledgements

We thank Elisa Cascardi, Jennifer Pak, and Pierre Louis Xech for their steadfast support throughout this project. We are grateful to Alexandre Gramfort, Arnaud Delorme, Bertrand Thirion, Christophe Pallier, Julie Boyle, Lune Bellec, Niall Holmes, Pierre Bourdillon, Stanislas Dehaene, Svetlana Pinet, Thomas Moreau, Valentin Wyart, and Yair Lakretz for their insightful discussions and continued feedback, and to the broader open-source neuroscience community for the tools and standards on which this work builds.

References

Alexandre Abraham, Fabian Pedregosa, Michael Eickenberg, Philippe Gervais, Andreas Mueller, Jean Kossaifi, Alexandre Gramfort, Bertrand Thirion, and Gaël Varoquaux. Machine learning for neuroimaging with scikit-learn. *Frontiers in Neuroinformatics*, 8:14, 2014.

Emily J Allen, Ghislain St-Yves, Yihan Wu, Jesse L Breedlove, Jacob S Prince, Logan T Dowdle, Matthias Nau, Brad Caron, Franco Pestilli, Ian Charest, et al. A massive 7T fMRI dataset to bridge cognitive neuroscience and artificial intelligence. *Nature neuroscience*, 25(1):116–126, 2022.

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In *Advances in Neural Information Processing Systems*, volume 33, pages 12449–12460, 2020.

Charlotte Caucheteux and Jean-Rémi King. Brains and algorithms partially converge in natural language processing. *Communications biology*, 5(1):134, 2022.

Alexandre Défossez, Charlotte Caucheteux, Jérémy Rapin, Ori Kabeli, and Jean-Rémi King. Decoding speech perception from non-invasive brain recordings. *Nature Machine Intelligence*, 5(10):1097–1107, 2023. doi: 10.1038/s42256-023-00714-5.

Arnaud Delorme and Scott Makeig. EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *Journal of Neuroscience Methods*, 134(1):9–21, 2004.

Oscar Esteban, Christopher J Markiewicz, Ross W Blair, Craig A Moodie, A Ilkay Isik, Asier Erramuzpe, James D Kent, Mathias Goncalves, Elizabeth DuPre, Madeleine Snyder, et al. fMRIPrep: a robust preprocessing pipeline for functional MRI. *Nature methods*, 16(1):111–116, 2019.

Krzysztof J Gorgolewski, Tibor Auer, Vince D Calhoun, R Cameron Craddock, Samir Das, Eugene P Duff, Guillaume Flandin, Satrajit S Ghosh, Tristan Glatard, Yaroslav O Halchenko, et al. The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. *Scientific Data*, 3(1):1–9, 2016.

Alexandre Gramfort, Martin Luessi, Eric Larson, Denis A Engemann, Daniel Strohmeier, Christian Brodbeck, Roman Goj, Mainak Jas, Teon Brooks, Lauri Parkkonen, and Matti S Hämäläinen. MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience*, 7:267, 2013.

Laura Gwilliams, Graham Flick, Alec Marantz, Liina Pylkkänen, David Poeppel, and Jean-Rémi King. Introducing MEG-MASC a high-quality magneto-encephalography dataset for evaluating natural speech processing. *Scientific data*, 10(1):862, 2023.

Uri Hasson, Yuval Nir, Ifat Levy, Galit Fuhrmann, and Rafael Malach. Intersubject synchronization of cortical activity during natural vision. *Science*, 303(5664):1634–1640, 2004.

Wes McKinney. Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56–61, 2010.

Samuel A Nastase, Yun-Fei Liu, Hanna Hillman, Asieh Zadbood, Liat Hasenfratz, Neggin Keshavarzian, Janice Chen, Christopher J Honey, Yaara Yeshurun, Mor Regev, et al. The "Narratives" fMRI dataset for evaluating models of naturalistic language comprehension. *Scientific data*, 8(1):250, 2021.

Robert Oostenveld, Pascal Fries, Eric Maris, and Jan-Mathijs Schoffelen. FieldTrip: Open source software for advanced analysis of MEG, EEG, and invasive electrophysiological data. *Computational Intelligence and Neuroscience*, 2011: 156869, 2011.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision, 2023.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Alec Radford, Jeff Wu, R. Child, D. Luan, Dario Amodei, and I. Sutskever. Language models are unsupervised multitask learners. OpenAI blog, 2019.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518, 2023.

J. Rapin and J.-R. King. Exca - Execution and caching. https://github.com/facebookresearch/exca, 2024.

Blake A Richards, Timothy P Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, et al. A deep learning framework for neuroscience. *Nature Neuroscience*, 22(11):1761–1770, 2019.

Robin Tibor Schirrmeister, Jost Tobias Springenberg, Lukas Dominique Josef Fiederer, Martin Glasstetter, Katharina Eggensperger, Michael Tangermann, Frank Hutter, Wolfram Burgard, and Tonio Ball. Deep learning with convolutional neural networks for EEG decoding and visualization. *Human Brain Mapping*, 38(11):5391–5420, 2017.

Martin Schrimpf, Idan Blank, Greta Tuckute, Carina Kauf, Eghbal A Hosseini, Nancy Kanwisher, Joshua Tenenbaum, and Evelina Fedorenko. Artificial neural networks accurately predict language processing in the brain. *BioRxiv*, pages 2020–06, 2020.

Saurabh Sonkusare, Michael Breakspear, and Christine Guo. Naturalistic stimuli in neuroscience: critically acclaimed. *Trends in Cognitive Sciences*, 23(8):699–714, 2019.

François Tadel, Sylvain Baillet, John C Mosher, Dimitrios Pantazis, and Richard M Leahy. Brainstorm: A user-friendly application for MEG/EEG analysis. *Computational Intelligence and Neuroscience*, 2011:879716, 2011.

Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. VideoMAE: Masked autoencoders are data-efficient learners for self-supervised video pre-training. In *Advances in Neural Information Processing Systems*, volume 35, pages 10078–10093, 2022.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models, 2023.

Ritchie Vink. Polars: Blazingly fast DataFrames in Rust, Python, Node.js, 2024. https://github.com/pola-rs/polars.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, 2020.

Daniel LK Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356–365, 2016.