```
pip install tensorflow keras matplotlib numpy pandas
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/
Requirement already satisfied: keras in /usr/local/lib/python3.12/dist-
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/pyth
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/p
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/py
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/pytho
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/pyth
Requirement already satisfied: packaging in /usr/local/lib/python3.12/d
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.2
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/py
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.12
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/pytho
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/l
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/py
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/py
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.1
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/li
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/pytho
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/pyth
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/pyth
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/d
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/pytho
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/p
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.1
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/py
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/l
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.1
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/py
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/py
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python
```

```
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/li
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/pyth
```

```
import numpy as np
import pandas as pd
import tensorflow as tf
import keras
from keras import layers
import matplotlib.pyplot as plt

# ============================================================
# 1. LOAD + PREP DATA
# ============================================================

# Expect health_data.csv in same directory
df = pd.read_csv("Multiset-Dataset.csv").dropna()

# Parse time for plotting, but don't use it as a feature
df["Date/Time"] = pd.to_datetime(df["Date/Time"])
timestamps = df["Date/Time"].astype("int64") // 10**9  # seconds since

# Keep only numeric columns as features (drop the datetime)
feature_df = df.select_dtypes(include=[np.number]).copy()

# Convert to numpy
X = feature_df.to_numpy().astype("float32")
num_features = X.shape[1]

print(f"Loaded data with shape: {X.shape} (samples, features)")

# ============================================================
# 2. CUSTOM SIREN SINE ACTIVATION
# ============================================================

@keras.utils.register_keras_serializable()
def sine_activation(x):
    return tf.sin(x)


# ============================================================
# 3. HYPERDIMENSIONAL PROJECTION (HDP)
# ============================================================
```

```
def HD_Projection(dim=4096):
    return keras.Sequential(
        [
            layers.Dense(dim, activation=sine_activation),
            layers.Dense(dim, activation=sine_activation),
            layers.Dense(dim, activation=None),
        ],
        name="HD_Proj",
    )

hd_proj = HD_Projection(dim=4096)


# ============================================================
# 4. ODE BLOCK + TRANSFORMER-LIKE LATENT DYNAMICS
# ============================================================

class ODEBlock(layers.Layer):
    def __init__(self, units, **kwargs):
        super().__init__(**kwargs)
        self.dense1 = layers.Dense(units, activation="tanh")
        self.dense2 = layers.Dense(units)

    def call(self, x):
        dx = self.dense2(self.dense1(x))
        # Small Euler integration step
        return x + 0.1 * dx


def EXO_Transformer(hidden=512, heads=8):
    inp = keras.Input((4096,), name="exo_tr_input")

    # Treat this as a 1-step "sequence" for attention
    x = layers.Reshape((1, 4096))(inp)

    # Multi-head self-attention over feature space
    x_attn = layers.MultiHeadAttention(num_heads=heads, key_dim=64)(
        x, x
    )
    x = layers.Add()([x, x_attn])
    x = layers.LayerNormalization()(x)

    # MLP block
    mlp = layers.Dense(hidden, activation="gelu")(x)
```

```
        mlp = layers.Dense(4096)(mlp)
        x = layers.Add()([x, mlp])

        # ODE dynamics in feature-latent space
        x = ODEBlock(4096, name="latent_ode")(x)

        # Flatten back to vector
        x = layers.Flatten()(x)

        return keras.Model(inp, x, name="EXO_Transformer")

    exo_tr = EXO_Transformer()


    # ============================================================
    # 5. CROSS-MODAL FUSION + SCALAR HEAD
    # ============================================================

    def FusionBlock():
        inp = keras.Input((4096,), name="fusion_input")
        x = layers.Dense(1024, activation="gelu")(inp)
        x = layers.Dense(256, activation="gelu")(x)
        x = layers.Dense(64, activation="gelu")(x)
        return keras.Model(inp, x, name="Fusion")

    fusion = FusionBlock()

    psi_head = keras.Sequential(
        [
            layers.Dense(32, activation="gelu"),
            layers.Dense(1, activation="sigmoid"),  # EXO-Ψ in [0, 1]
        ],
        name="EXO_Psi_Head",
    )


    # ============================================================
    # 6. EXO-HYPERMIND MODEL (SUBCLASSED, SELF-SUPERVISED)
    # ============================================================

    class EXO_HYPERMIND(keras.Model):
        def __init__(self, proj, trans, fuse, psi, **kwargs):
            super().__init__(**kwargs)
            self.proj = proj
            self.trans = trans
```

```
        self.fuse = fuse
        self.psi = psi

    def compile(self, optimizer, **kwargs):
        super().compile(**kwargs)
        self.optimizer = optimizer

    def train_step(self, data):
        # Keras may pass (x, y) even if we don't give y; handle robust
        if isinstance(data, tuple) or isinstance(data, list):
            x = data[0]
        else:
            x = data

        with tf.GradientTape() as tape:
            # Hyperdimensional projection
            hd = self.proj(x, training=True)

            # Transformer-like latent dynamics
            z = self.trans(hd, training=True)

            # Fusion head
            fused = self.fuse(z, training=True)

            # Scalar EXO-Ψ prediction
            psi = self.psi(fused, training=True)

            # -------- Self-supervised temporal-like losses within ba
            batch_size = tf.shape(x)[0]

            def compute_smooth_and_chaos():
                # Smoothness in input space (encourage continuity)
                smooth = tf.reduce_mean(tf.square(x[1:] - x[:-1]))

                # Chaos / sensitivity in latent space (encourage expre
                chaos = tf.reduce_mean(tf.abs(z[1:] - z[:-1]))
                return smooth, chaos

            smooth, chaos = tf.cond(
                tf.greater(batch_size, 1),
                lambda: compute_smooth_and_chaos(),
                lambda: (tf.constant(0.0, dtype=tf.float32),
                         tf.constant(0.0, dtype=tf.float32)),
            )
```

```
            # Encourage non-trivial psi distribution
            psi_mean = tf.reduce_mean(psi)

            # Total loss: chaos (want some), smoothness (regularize),
            loss = chaos + 0.1 * smooth + 0.5 * psi_mean

        grads = tape.gradient(loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weigl

        return {
            "loss": loss,
            "smooth_loss": smooth,
            "chaos_loss": chaos,
            "psi_mean": psi_mean,
        }

    def call(self, x, training=False):
        hd = self.proj(x, training=training)
        z = self.trans(hd, training=training)
        fused = self.fuse(z, training=training)
        return self.psi(fused, training=training)


# ============================================================
# 7. INSTANTIATE + TRAIN
# ============================================================

model = EXO_HYPERMIND(hd_proj, exo_tr, fusion, psi_head)
model.compile(optimizer=keras.optimizers.Adam(1e-4))

print("\n🔥 TRAINING EXO-HYPERMIND...\n")
model.fit(X, epochs=25, batch_size=32, verbose=1)


# ============================================================
# 8. COMPUTE EXO-Ψ INDEX (0–100)
# ============================================================

exo_raw = model.predict(X).flatten()
EXO_PSI = 100.0 * (exo_raw - exo_raw.min()) / (exo_raw.max() - exo_ra

df["EXO_PSI"] = EXO_PSI

print("\nSample of EXO_PSI values:")
print(df["EXO_PSI"].head())
```

```
# ============================================================
# 9. PLOT EXO-Ψ OVER TIME
# ============================================================

plt.figure(figsize=(16, 7))
plt.plot(timestamps, EXO_PSI, label="EXO-Ψ Index", linewidth=3, color=
plt.title("EXO-Ψ: Hyperdimensional Transformer Human Stability Index")
plt.xlabel("Time")
plt.ylabel("Index (0–100)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

```
Loaded data with shape: (49887, 42) (samples, features)

🔥 TRAINING EXO-HYPERMIND...

Epoch 1/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 49s 18ms/step - chaos_loss: 0.1032 - los
Epoch 2/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0133 - loss
Epoch 3/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0096 - loss
Epoch 4/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0071 - loss
Epoch 5/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0057 - loss
Epoch 6/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0049 - loss
Epoch 7/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0043 - loss
Epoch 8/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0044 - loss
Epoch 9/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0038 - loss
Epoch 10/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0035 - loss
Epoch 11/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0034 - loss
Epoch 12/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0032 - loss
Epoch 13/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0030 - loss
Epoch 14/25
```

```
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0031 - loss
Epoch 15/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0026 - loss
Epoch 16/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0028 - loss
Epoch 17/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0027 - loss
Epoch 18/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0028 - loss
Epoch 19/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0029 - loss
Epoch 20/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0022 - loss
Epoch 21/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0025 - loss
Epoch 22/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0023 - loss
Epoch 23/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0024 - loss
Epoch 24/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0022 - loss
Epoch 25/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0024 - loss
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 8s 2ms/step

Sample of EXO_PSI values:
0    1.541314e-14
1    1.535885e-14
```

```
"""
EXO-Hypermind v2.0
Hyperdimensional Self-Supervised Human Stability Index (EXO-Ψ)

Requirements:
    pip install numpy pandas scikit-learn matplotlib tensorflow
"""

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

# ------------------------------------------------------------
```

```
# 1. Load & preprocess data
# ------------------------------------------------------------

CSV_PATH = "Multiset-Dataset.csv"   # <-- change if needed
TIME_COL = "Date/Time"

print("📥 Loading data...")
df = pd.read_csv(CSV_PATH, parse_dates=[TIME_COL])

# Keep a copy of the time axis
time_values = df[TIME_COL].values

# Drop non-numeric / time column
df_num = df.drop(columns=[TIME_COL])

# Ensure all columns are numeric
df_num = df_num.apply(pd.to_numeric, errors="coerce")

# Handle missing values (forward fill, then back fill if needed)
df_num = df_num.fillna(method="ffill").fillna(method="bfill")

X = df_num.values.astype("float32")
n_samples, n_features = X.shape
print(f"Loaded data with shape: ({n_samples}, {n_features})  (samples,

# Scale features for stable training
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X).astype("float32")

# ------------------------------------------------------------
# 2. Custom layers
# ------------------------------------------------------------

class SineLayer(layers.Layer):
    """Simple SIREN-style sine activation as a layer."""
    def call(self, inputs):
        return tf.math.sin(inputs)

    def get_config(self):
        base = super().get_config()
        return base

class Sampling(layers.Layer):
    """Reparameterization trick: z = μ + σ * ε"""
    def call(self, inputs):
```

```
        z_mean, z_logvar = inputs
        eps = tf.random.normal(shape=tf.shape(z_mean))
        return z_mean + tf.exp(0.5 * z_logvar) * eps

# ------------------------------------------------------
# 3. Build Encoder, Decoder, and ψ-head
# ------------------------------------------------------

latent_dim = 8          # low-dim chaotic manifold
hd_dim = 1024           # hyperdimensional embedding size

# ----- Encoder -----
encoder_inputs = keras.Input(shape=(n_features,), name="encoder_input"

x = layers.Dense(256, activation="gelu")(encoder_inputs)
x = layers.Dense(512, activation="gelu")(x)

# Hyperdimensional projection
hd = layers.Dense(hd_dim)(x)
hd = SineLayer()(hd)           # SIREN-style nonlinearity
hd = layers.LayerNormalization()(hd)

# Latent statistics
z_mean = layers.Dense(latent_dim, name="z_mean")(hd)
z_logvar = layers.Dense(latent_dim, name="z_logvar")(hd)
z = Sampling()([z_mean, z_logvar])

encoder = keras.Model(
    encoder_inputs, [z_mean, z_logvar, z, hd],
    name="exo_encoder"
)

# ----- Decoder -----
latent_inputs = keras.Input(shape=(latent_dim,), name="decoder_input"
x = layers.Dense(512, activation="gelu")(latent_inputs)
x = layers.Dense(256, activation="gelu")(x)
decoder_outputs = layers.Dense(n_features, name="decoder_output")(x)

decoder = keras.Model(latent_inputs, decoder_outputs, name="exo_decod

# ----- ψ-head (stability index head) -----
psi_inputs = keras.Input(shape=(hd_dim,), name="psi_input")
p = layers.Dense(128, activation="gelu")(psi_inputs)
p = layers.Dense(64, activation="gelu")(p)
psi_output = layers.Dense(1, activation=None, name="psi_raw")(p)
```

```
psi_head = keras.Model(psi_inputs, psi_output, name="psi_head")

# ------------------------------------------------------------
# 4. EXO-Hypermind model (custom training loop)
# ------------------------------------------------------------

class ExoHypermind(keras.Model):
    """
    Self-supervised model that:
      - Reconstructs X (VAE style)
      - Encourages smoothness in HD space
      - Learns a scalar ψ to predict latent chaos between steps
    """
    def __init__(self, encoder, decoder, psi_head, **kwargs):
        super().__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.psi_head = psi_head

        # trackers
        self.total_loss_tracker = keras.metrics.Mean(name="total_loss'
        self.recon_loss_tracker = keras.metrics.Mean(name="recon_loss'
        self.smooth_loss_tracker = keras.metrics.Mean(name="smooth_los
        self.psi_loss_tracker = keras.metrics.Mean(name="psi_loss")
        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss")

    @property
    def metrics(self):
        return [
            self.total_loss_tracker,
            self.recon_loss_tracker,
            self.smooth_loss_tracker,
            self.psi_loss_tracker,
            self.kl_loss_tracker,
        ]

    def train_step(self, data):
        # data is just X; ignore y
        x = data

        with tf.GradientTape() as tape:
            z_mean, z_logvar, z, hd = self.encoder(x, training=True)
            x_recon = self.decoder(z, training=True)
```

```
            # 1) Reconstruction loss (MSE across features)
            recon_error = tf.reduce_sum(tf.square(x - x_recon), axis=1
            recon_loss = tf.reduce_mean(recon_error)

            # 2) KL divergence (VAE regularizer)
            kl_term = -0.5 * tf.reduce_sum(
                1 + z_logvar - tf.square(z_mean) - tf.exp(z_logvar), a
            )
            kl_loss = tf.reduce_mean(kl_term)

            # 3) Smoothness in HD space: penalize big jumps hd[t+1]-hd
            hd_diff = hd[1:] - hd[:-1]
            smooth_loss = tf.reduce_mean(tf.square(hd_diff))

            # 4) Latent chaos target from z
            z_diff = z[1:] - z[:-1]
            chaos_target = tf.reduce_mean(tf.abs(z_diff), axis=1, keep

            # ψ prediction
            psi_pred = self.psi_head(hd[:-1], training=True)   # align
            psi_loss = tf.reduce_mean(tf.square(psi_pred - chaos_targe

            # 5) Total loss with scaling
            # Note: smooth_loss and psi_loss are already small due to
            total_loss = (
                recon_loss
                + 0.1 * smooth_loss
                + 0.1 * psi_loss
                + 0.01 * kl_loss
            )

        grads = tape.gradient(total_loss, self.trainable_variables)
        self.optimizer.apply_gradients(zip(grads, self.trainable_varia

        self.total_loss_tracker.update_state(total_loss)
        self.recon_loss_tracker.update_state(recon_loss)
        self.smooth_loss_tracker.update_state(smooth_loss)
        self.psi_loss_tracker.update_state(psi_loss)
        self.kl_loss_tracker.update_state(kl_loss)

        return {
            "loss": self.total_loss_tracker.result(),
            "recon_loss": self.recon_loss_tracker.result(),
            "smooth_loss": self.smooth_loss_tracker.result(),
            "psi_loss": self.psi_loss_tracker.result(),
```

```
            "kl_loss": self.kl_loss_tracker.result(),
        }

    def call(self, inputs, training=False):
        z_mean, z_logvar, z, hd = self.encoder(inputs, training=train
        x_recon = self.decoder(z, training=training)
        psi_pred = self.psi_head(hd, training=training)
        return x_recon, psi_pred, z_mean, z_logvar, z, hd


# ------------------------------------------------------------
# 5. Instantiate & train model
# ------------------------------------------------------------

tf.random.set_seed(42)
np.random.seed(42)

model = ExoHypermind(encoder, decoder, psi_head, name="exo_hypermind"

optimizer = keras.optimizers.Adam(learning_rate=1e-3)
model.compile(optimizer=optimizer)

EPOCHS = 25
BATCH_SIZE = 32

print("\n🔥 TRAINING EXO-HYPERMIND v2.0...\n")

history = model.fit(
    X_scaled,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    shuffle=False,   # keep temporal order for smoothness & chaos tar
    verbose=1,
)

# ------------------------------------------------------------
# 6. Compute EXO-Ψ index over full dataset
# ------------------------------------------------------------

print("\n🧠 Computing EXO-Ψ index...")

# Run full forward pass
x_recon, psi_pred, z_mean, z_logvar, z, hd = model(X_scaled, training=

psi_raw = psi_pred.numpy().reshape(-1)
```

```
# Stability is derived from chaos in latent z as well
z_np = z.numpy()
z_diff = np.abs(z_np[1:] - z_np[:-1]).mean(axis=1)
z_diff = np.insert(z_diff, 0, z_diff[0])  # pad to same length

# Combine ψ-head output and latent chaos
combined = 0.7 * psi_raw + 0.3 * z_diff

# Normalize to 0–100
min_val = combined.min()
max_val = combined.max()
exo_psi_0_100 = 100.0 * (combined - min_val) / (max_val - min_val + 1e

exo_series = pd.Series(exo_psi_0_100, index=pd.to_datetime(time_values

print(exo_series.head())

# ------------------------------------------------------------
# 7. Plot EXO-Ψ vs time
# ------------------------------------------------------------

plt.figure(figsize=(14, 5))
plt.plot(exo_series.index, exo_series.values, label="EXO-Ψ Index", lin
plt.title("EXO-Ψ: Hyperdimensional Human Stability Index (0–100)")
plt.xlabel("Time")
plt.ylabel("Index (0–100)")
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# ------------------------------------------------------------
# 8. Optional: compare with a few raw metrics
# ------------------------------------------------------------

# Example: compare with screen_on (if exists) and heart rate avg
cols_to_compare = []
if "screen_on" in df.columns:
    cols_to_compare.append("screen_on")
if "Heart Rate [Avg] (count/min)" in df.columns:
    cols_to_compare.append("Heart Rate [Avg] (count/min)")
if "Sleep Analysis [Total] (hr)" in df.columns:
    cols_to_compare.append("Sleep Analysis [Total] (hr)")
```

```
if cols_to_compare:
    fig, ax1 = plt.subplots(figsize=(14, 5))
    ax1.plot(exo_series.index, exo_series.values, color="tab:purple",
             label="EXO-Ψ Index", linewidth=1.2)
    ax1.set_ylabel("EXO-Ψ (0–100)", color="tab:purple")
    ax1.tick_params(axis="y", labelcolor="tab:purple")

    ax2 = ax1.twinx()
    for col in cols_to_compare:
        ax2.plot(df[TIME_COL], df[col], label=col, alpha=0.6)
    ax2.set_ylabel("Raw metrics")
    ax2.tick_params(axis="y")

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    ax1.legend(lines1 + lines2, labels1 + labels2, loc="upper right")

    plt.title("EXO-Ψ vs Screen Time / Heart Rate / Sleep")
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()
else:
    print("\n(ℹ️ Skipping comparison plot — expected columns not foun
```

```
📥 Loading data...
/tmp/ipython-input-1546121450.py:39: FutureWarning: DataFrame.fillna wi
  df_num = df_num.fillna(method="ffill").fillna(method="bfill")
Loaded data with shape: (49887, 42)  (samples, features)

🔥 TRAINING EXO-HYPERMIND v2.0...

Epoch 1/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 19s 6ms/step - kl_loss: 44.3173 - loss:
Epoch 2/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 39.6211 - loss: 2
Epoch 3/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 44.4612 - loss: 2
Epoch 4/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 40.7254 - loss: 2
Epoch 5/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 39.3428 - loss: 2
Epoch 6/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 623.9925 - loss:
Epoch 7/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 31.3595 - loss: 2
Epoch 8/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 30.9201 - loss: 1
```

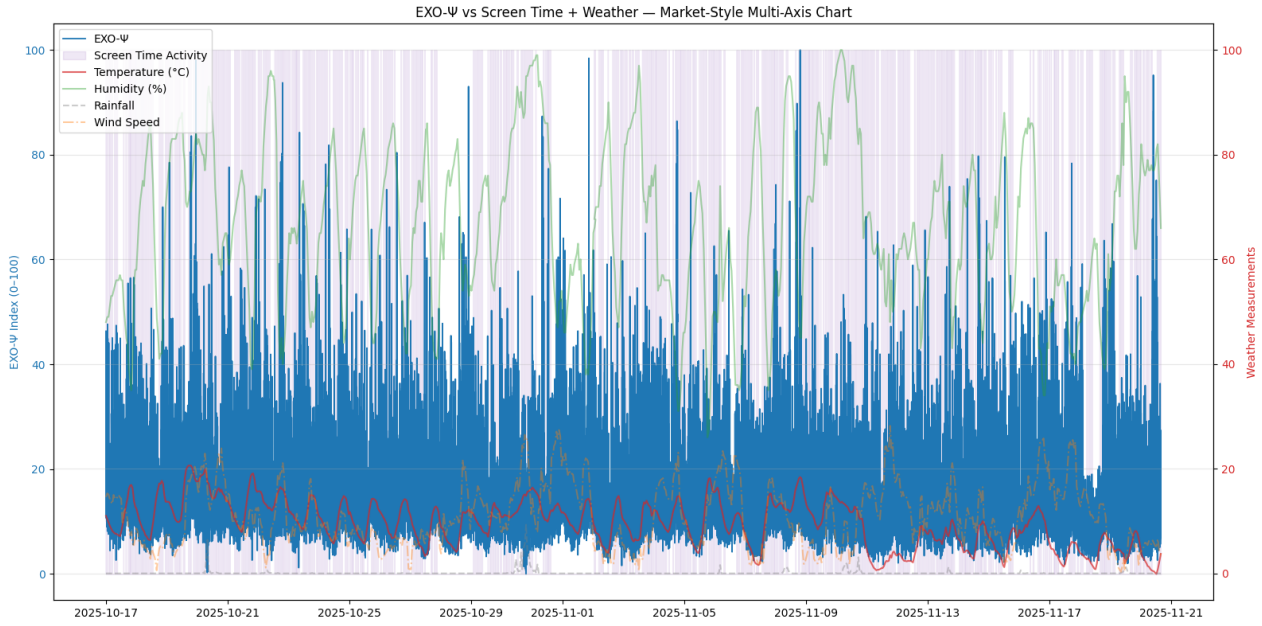
```
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 30.9201 - loss: 2
Epoch 9/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 31.8939 - loss: 2
Epoch 10/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 32.8737 - loss: 2
Epoch 11/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 33.7272 - loss: 2
Epoch 12/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 34.2173 - loss: 2
Epoch 13/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 35.6504 - loss: 2
Epoch 14/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 36.7450 - loss: 2
Epoch 15/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 37.1175 - loss: 2
Epoch 16/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 37.6075 - loss: 2
Epoch 17/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 37.1940 - loss: 2
Epoch 18/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 38.4412 - loss: 2
Epoch 19/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 38.2556 - loss: 2
Epoch 20/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 36.1621 - loss: 2
Epoch 21/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 35.2592 - loss: 2
Epoch 22/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 34.6613 - loss: 2
Epoch 23/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 33.1588 - loss: 2
Epoch 24/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 31.9889 - loss: 1
Epoch 25/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - kl_loss: 30.9026 - loss: 1

🧠 Computing EXO-Ψ index...
```

```
# ------------------------------------------------------------
# 9. STOCK-MARKET STYLE MULTI-AXIS COMPARISON CHART
# ------------------------------------------------------------

print("\n📊 Plotting multi-axis EXO-Ψ vs Screen Time & Weather...")

# Choose which external factors to overlay
screen_col = "screen_on" if "screen_on" in df.columns else None
temp_col = "temperature" if "temperature" in df.columns else None
humid_col = "humidity" if "humidity" in df.columns else None
rain_col = "rain" if "rain" in df.columns else None
```

```
wind_col = "wind_speed" if "wind_speed" in df.columns else None

plt.figure(figsize=(16, 8))

# --- Main axis (EXO-Ψ index) ---
ax1 = plt.gca()
ax1.plot(exo_series.index, exo_series.values,
         color="tab:blue", linewidth=1.3, label="EXO-Ψ")

ax1.set_ylabel("EXO-Ψ Index (0–100)", color="tab:blue")
ax1.tick_params(axis="y", labelcolor="tab:blue")

# --- Screen-time AREA CHART ---
if screen_col:
    ax1.fill_between(
        df[TIME_COL],
        df[screen_col] * exo_series.max(),  # scale area visual
        alpha=0.15,
        color="tab:purple",
        label="Screen Time Activity",
        step="pre"
    )


# --- Secondary Y-axis for weather ---
ax2 = ax1.twinx()

if temp_col:
    ax2.plot(df[TIME_COL], df[temp_col], color="tab:red",
             label="Temperature (°C)", alpha=0.7)

if humid_col:
    ax2.plot(df[TIME_COL], df[humid_col], color="tab:green",
             label="Humidity (%)", alpha=0.4)

if rain_col:
    ax2.plot(df[TIME_COL], df[rain_col], color="tab:gray",
             label="Rainfall", alpha=0.4, linestyle="--")

if wind_col:
    ax2.plot(df[TIME_COL], df[wind_col], color="tab:orange",
             label="Wind Speed", alpha=0.4, linestyle="-.")

ax2.set_ylabel("Weather Measurements", color="tab:red")
ax2.tick_params(axis="y", labelcolor="tab:red")
```

```
# --- Combined Legend (both axes) ---
lines1, labels1 = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()

plt.legend(lines1 + lines2, labels1 + labels2, loc="upper left")

plt.title("EXO-Ψ vs Screen Time + Weather — Market-Style Multi-Axis Cl
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

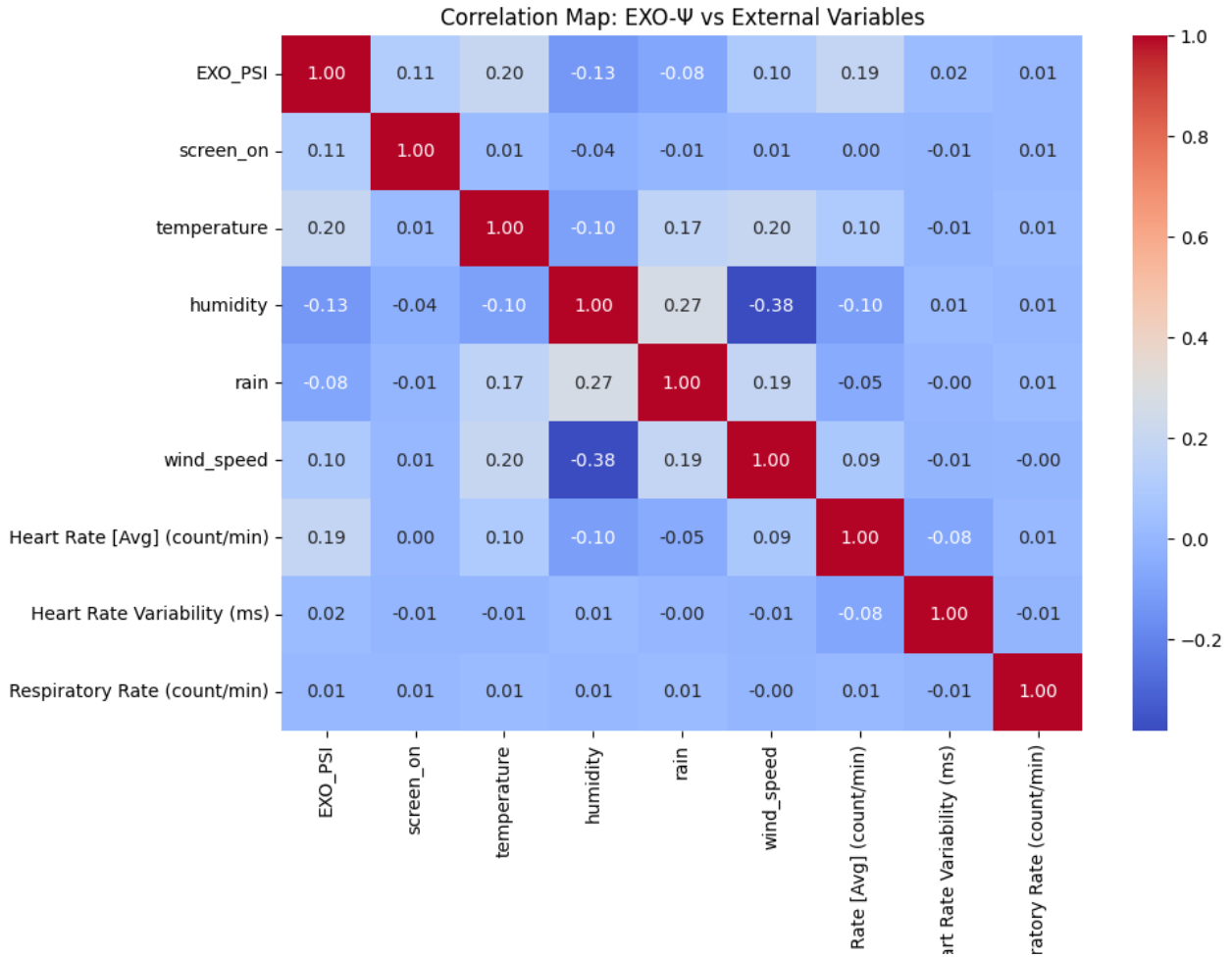
```
Plotting multi-axis EXO-Ψ vs Screen Time & Weather...
```

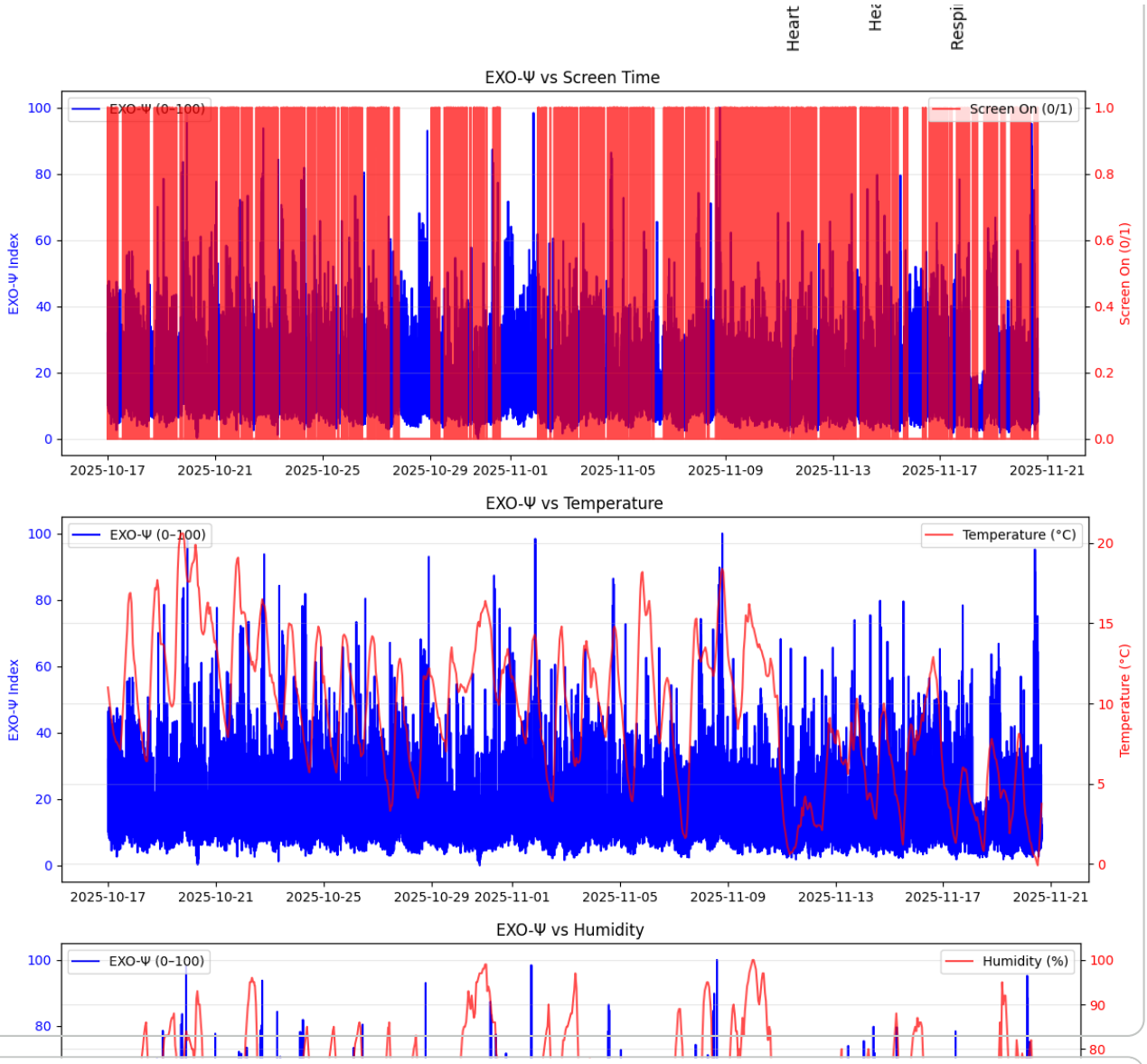


```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# ------------------------------------------------------------
# 1. CLEAN CORRELATION HEATMAP (EXO-Ψ vs external metrics)
# ------------------------------------------------------------

corr_df = df.copy()
corr_df["EXO_PSI"] = exo_series.values

corr = corr_df[["EXO_PSI", "screen_on", "temperature",
                "humidity", "rain", "wind_speed",
                "Heart Rate [Avg] (count/min)",
                "Heart Rate Variability (ms)",
                "Respiratory Rate (count/min)"
               ]].corr()

plt.figure(figsize=(10,7))
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Map: EXO-Ψ vs External Variables")
plt.show()


# ------------------------------------------------------------
# 2. CLEAN 2-AXIS RELATION GRAPH (one factor at a time)
# ------------------------------------------------------------

def plot_relation(y, y_label, title):
    plt.figure(figsize=(14,5))

    ax1 = plt.gca()
    ax1.plot(df[TIME_COL], exo_series, color='blue', label="EXO-Ψ (0-1
    ax1.set_ylabel("EXO-Ψ Index", color="blue")
    ax1.tick_params(axis="y", labelcolor="blue")

    ax2 = ax1.twinx()
```

```
    ax2.plot(df[TIME_COL], df[y], color='red', alpha=0.7, label=y_lab
    ax2.set_ylabel(y_label, color="red")
    ax2.tick_params(axis="y", labelcolor="red")

    plt.title(title)
    plt.grid(True, alpha=0.3)
    ax1.legend(loc="upper left")
    ax2.legend(loc="upper right")

    plt.show()


plot_relation("screen_on", "Screen On (0/1)",
              "EXO-Ψ vs Screen Time")

plot_relation("temperature", "Temperature (°C)",
              "EXO-Ψ vs Temperature")

plot_relation("humidity", "Humidity (%)",
              "EXO-Ψ vs Humidity")

plot_relation("wind_speed", "Wind Speed",
              "EXO-Ψ vs Wind Speed")


# ------------------------------------------------------------
# 3. SCATTER PLOT + TREND LINE
# ------------------------------------------------------------

def scatter_relation(x, x_label):
    plt.figure(figsize=(7,5))
    sns.regplot(x=df[x], y=exo_series, scatter_kws={'alpha':0.3})
    plt.xlabel(x_label)
    plt.ylabel("EXO-Ψ Index (0–100)")
    plt.title(f"Scatter Relationship: EXO-Ψ vs {x_label}")
    plt.grid(True, alpha=0.3)
    plt.show()

scatter_relation("screen_on", "Screen Time")
scatter_relation("temperature", "Temperature (°C)")
scatter_relation("humidity", "Humidity (%)")
scatter_relation("Heart Rate [Avg] (count/min)", "Heart Rate (Avg)")


# ------------------------------------------------------------
```

```
# 4. LAG ANALYSIS — CAUSE & EFFECT
# ------------------------------------------------------------

lags = range(-6, 7)  # -6h to +6h
lag_corr = []

for lag in lags:
    shifted = exo_series.shift(lag)
    c = shifted.corr(df["screen_on"])
    lag_corr.append(c)

plt.figure(figsize=(10,5))
plt.plot(lags, lag_corr, marker='o')
plt.axhline(0, color='black', linewidth=1)
plt.title("Lag Correlation: Screen Time → EXO-Ψ Impact")
plt.xlabel("Lag (hours)")
plt.ylabel("Correlation")
plt.grid(True)
plt.show()
```






```
import numpy as np
import pandas as pd
import tensorflow as tf
import keras
from keras import layers
import matplotlib.pyplot as plt

# ============================================================
# 1. LOAD + PREP DATA
# ============================================================

df = pd.read_csv("Multiset-Dataset.csv").dropna()

df["Date/Time"] = pd.to_datetime(df["Date/Time"])
timestamps = df["Date/Time"].astype("int64") // 10**9  # seconds since
```

```
feature_df = df.select_dtypes(include=[np.number]).copy()
X = feature_df.to_numpy().astype("float32")
num_features = X.shape[1]

print(f"Loaded data with shape: {X.shape} (samples, features)")

# ============================================================
# 2. CUSTOM SIREN SINE ACTIVATION
# ============================================================

@keras.utils.register_keras_serializable()
def sine_activation(x):
    return tf.sin(x)

# ============================================================
# 3. HYPERDIMENSIONAL PROJECTION (HDP)
# ============================================================

def HD_Projection(dim=4096):
    return keras.Sequential(
        [
            layers.Dense(dim, activation=sine_activation),
            layers.Dense(dim, activation=sine_activation),
            layers.Dense(dim, activation=None),
        ],
        name="HD_Proj",
    )

hd_proj = HD_Projection(dim=4096)

# ============================================================
# 4. ODE BLOCK + TRANSFORMER DYNAMICS
# ============================================================

class ODEBlock(layers.Layer):
    def __init__(self, units, **kwargs):
        super().__init__(**kwargs)
        self.dense1 = layers.Dense(units, activation="tanh")
        self.dense2 = layers.Dense(units)

    def call(self, x):
        dx = self.dense2(self.dense1(x))
        return x + 0.1 * dx
```

```
def EXO_Transformer(hidden=512, heads=8):
    inp = keras.Input((4096,), name="exo_tr_input")
    x = layers.Reshape((1, 4096))(inp)
    x_attn = layers.MultiHeadAttention(num_heads=heads, key_dim=64)(x,
    x = layers.Add()([x, x_attn])
    x = layers.LayerNormalization()(x)
    mlp = layers.Dense(hidden, activation="gelu")(x)
    mlp = layers.Dense(4096)(mlp)
    x = layers.Add()([x, mlp])
    x = ODEBlock(4096, name="latent_ode")(x)
    x = layers.Flatten()(x)
    return keras.Model(inp, x, name="EXO_Transformer")

exo_tr = EXO_Transformer()

# ============================================================
# 5. CROSS-MODAL FUSION + PSI HEAD
# ============================================================

def FusionBlock():
    inp = keras.Input((4096,), name="fusion_input")
    x = layers.Dense(1024, activation="gelu")(inp)
    x = layers.Dense(256, activation="gelu")(x)
    x = layers.Dense(64, activation="gelu")(x)
    return keras.Model(inp, x, name="Fusion")

fusion = FusionBlock()

psi_head = keras.Sequential(
    [
        layers.Dense(32, activation="gelu"),
        layers.Dense(1, activation="sigmoid"),
    ],
    name="EXO_Psi_Head",
)

# ============================================================
# 6. EXO-HYPERMIND MODEL
# ============================================================

class EXO_HYPERMIND(keras.Model):
    def __init__(self, proj, trans, fuse, psi, **kwargs):
        super().__init__(**kwargs)
        self.proj = proj
        self.trans = trans
```

```
            self.fuse = fuse
            self.psi = psi

        def compile(self, optimizer, **kwargs):
            super().compile(**kwargs)
            self.optimizer = optimizer

        def train_step(self, data):
            x = data[0] if isinstance(data, (tuple, list)) else data

            with tf.GradientTape() as tape:
                hd = self.proj(x, training=True)
                z = self.trans(hd, training=True)
                fused = self.fuse(z, training=True)
                psi = self.psi(fused, training=True)

                batch_size = tf.shape(x)[0]

                def compute_smooth_and_chaos():
                    smooth = tf.reduce_mean(tf.square(x[1:] - x[:-1]))
                    chaos = tf.reduce_mean(tf.abs(z[1:] - z[:-1]))
                    return smooth, chaos

                smooth, chaos = tf.cond(
                    tf.greater(batch_size, 1),
                    compute_smooth_and_chaos,
                    lambda: (0.0, 0.0)
                )

                psi_mean = tf.reduce_mean(psi)
                loss = chaos + 0.1 * smooth + 0.5 * psi_mean

            grads = tape.gradient(loss, self.trainable_weights)
            self.optimizer.apply_gradients(zip(grads, self.trainable_weigl

            return {"loss": loss, "smooth_loss": smooth,
                    "chaos_loss": chaos, "psi_mean": psi_mean}

        def call(self, x, training=False):
            hd = self.proj(x, training=training)
            z = self.trans(hd, training=training)
            fused = self.fuse(z, training=training)
            return self.psi(fused, training=training)

    # ============================================================
```

```
# 7. TRAIN THE MODEL
# ============================================================

model = EXO_HYPERMIND(hd_proj, exo_tr, fusion, psi_head)
model.compile(optimizer=keras.optimizers.Adam(1e-4))

print("\n🔥 TRAINING EXO-HYPERMIND...\n")
model.fit(X, epochs=25, batch_size=32, verbose=1)

# ============================================================
# 8. BUILD EXO-Ψ INDEX
# ============================================================

exo_raw = model.predict(X).flatten()
EXO_PSI = 100.0 * (exo_raw - exo_raw.min()) / (exo_raw.max() - exo_ra
df["EXO_PSI"] = EXO_PSI

print("\nSample of EXO_PSI values:")
print(df["EXO_PSI"].head())

# ============================================================
# 9. BASIC EXO-Ψ LINE CHART
# ============================================================

plt.figure(figsize=(16, 6))
plt.plot(df["Date/Time"], EXO_PSI, label="EXO-Ψ Index", linewidth=2.5
plt.title("EXO-Ψ: Hyperdimensional Human Stability Index")
plt.xlabel("Time")
plt.ylabel("Index (0-100)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# ============================================================
# 10. FULL QUANT MULTI-PANEL DASHBOARD
# ============================================================

def normalize(s):
    return 100 * (s - s.min()) / (s.max() - s.min() + 1e-9)

# Main series
exo_series = normalize(df["EXO_PSI"])

# Normalize external signals
```

```
screen = normalize(df["screen_on"]) if "screen_on" in df else df["EXO_
temp   = normalize(df["temperature"]) if "temperature" in df else exo_
humid  = normalize(df["humidity"]) if "humidity" in df else exo_series
rain   = normalize(df["rain"]) if "rain" in df else exo_series*0
wind   = normalize(df["wind_speed"]) if "wind_speed" in df else exo_se

# Composite driver indices
weather_idx = 0.4*temp + 0.3*humid + 0.2*wind + 0.1*rain
physio_idx = normalize(df.get("Heart Rate [Avg] (count/min)", 0)) * 0.
             normalize(df.get("Heart Rate Variability (ms)", 0)) * 0.3
             normalize(df.get("Respiratory Rate (count/min)", 0)) * 0.

# Rolling correlation (Screen vs EXO)
roll_corr = exo_series.rolling(240).corr(screen)

# ========== DASHBOARD PLOTTING ==========

fig, axes = plt.subplots(4, 1, figsize=(20, 16), sharex=True)

# Panel 1 — EXO-Ψ with EMAs
axes[0].plot(df["Date/Time"], exo_series, label="EXO-Ψ", color="purple
axes[0].plot(df["Date/Time"], exo_series.ewm(span=60).mean(), label="E
axes[0].plot(df["Date/Time"], exo_series.ewm(span=360).mean(), label="
axes[0].set_title("EXO-Ψ Quant Dashboard — Stability Regime View", for
axes[0].legend()
axes[0].grid(True)

# Panel 2 — Screen activity + correlation
axes[1].plot(df["Date/Time"], screen, label="Screen Activity", color="
ax2b = axes[1].twinx()
ax2b.plot(df["Date/Time"], roll_corr, label="Rolling Corr(EXO, Screen)
axes[1].grid(True)
axes[1].set_ylabel("Screen (0–100)")
axes[1].legend(loc="upper left")
ax2b.set_ylim(-1, 1)
ax2b.axhline(0, linestyle="--", color="gray")

# Panel 3 — Weather & Physio indices
axes[2].plot(df["Date/Time"], weather_idx, label="Weather Composite",
axes[2].plot(df["Date/Time"], physio_idx, label="Physio Composite", co
axes[2].legend()
axes[2].grid(True)

# Panel 4 — Correlation heatmap
corr_df = pd.DataFrame({
```

```
    "EXO": exo_series, "Screen": screen, "Temp": temp,
    "Humidity": humid, "Rain": rain, "Wind": wind,
    "WeatherIdx": weather_idx, "PhysioIdx": physio_idx
})
corr = corr_df.corr()

im = axes[3].imshow(corr, cmap="coolwarm", vmin=-1, vmax=1)
axes[3].set_xticks(range(len(corr.columns)))
axes[3].set_yticks(range(len(corr.columns)))
axes[3].set_xticklabels(corr.columns, rotation=45)
axes[3].set_yticklabels(corr.columns)
axes[3].set_title("Correlation Matrix")

for i in range(len(corr)):
    for j in range(len(corr)):
        axes[3].text(j, i, f"{corr.iloc[i, j]:.2f}", ha="center",
                     color="white" if abs(corr.iloc[i, j]) > 0.5 else '

fig.colorbar(im, ax=axes[3])
plt.tight_layout()
plt.show()
```

```
Loaded data with shape: (49887, 42) (samples, features)

🔥 TRAINING EXO-HYPERMIND...

Epoch 1/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 28s 11ms/step - chaos_loss: 0.1056 - los
Epoch 2/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0138 - loss
Epoch 3/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0095 - loss
Epoch 4/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0072 - loss
Epoch 5/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0057 - loss
Epoch 6/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0048 - loss
Epoch 7/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0044 - loss
Epoch 8/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0043 - loss
Epoch 9/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0038 - loss
Epoch 10/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0040 - loss
Epoch 11/25
```

```
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0036 - loss
Epoch 12/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0030 - loss
Epoch 13/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0031 - loss
Epoch 14/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0030 - loss
Epoch 15/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0032 - loss
Epoch 16/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0026 - loss
Epoch 17/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0029 - loss
Epoch 18/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0026 - loss
Epoch 19/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0024 - loss
Epoch 20/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0022 - loss
Epoch 21/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0031 - loss
Epoch 22/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0022 - loss
Epoch 23/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0022 - loss
Epoch 24/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0026 - loss
Epoch 25/25
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: 0.0022 - loss
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step

Sample of EXO_PSI values:
0    0.000054
1    0.000054
```

```
import numpy as np
import pandas as pd
import tensorflow as tf
import keras
from keras import layers
import matplotlib.pyplot as plt

# ============================================================
# 1. LOAD + PREP DATA
# ============================================================

# Expect Multiset-Dataset.csv in same directory
df_raw = pd.read_csv("Multiset-Dataset.csv").dropna(subset=["Date/Tim
```

```
# Parse time
df_raw["Date/Time"] = pd.to_datetime(df_raw["Date/Time"])

# Sort + dedupe + resample to smooth irregular sampling
df = (
    df_raw.sort_values("Date/Time")
          .drop_duplicates("Date/Time")
          .set_index("Date/Time")
          .resample("1min")
          .mean()
          .interpolate(limit_direction="both")
          .reset_index()
)

# Keep only numeric columns as features (drop datetime)
feature_df = df.select_dtypes(include=[np.number]).copy()
feature_cols = feature_df.columns.tolist()

# Full feature matrix
X_full = feature_df.to_numpy().astype("float32")
num_features = X_full.shape[1]

# Build forecasting pairs: (x_t -> x_{t+1})
X_t = X_full[:-1]
Y_future = X_full[1:]
time_t = df["Date/Time"].iloc[:-1].reset_index(drop=True)

print(f"Loaded data with shape: {X_full.shape} (samples, features)")
print(f"Training pairs: {X_t.shape} -> {Y_future.shape}")

# ============================================================
# 2. CUSTOM SIREN SINE ACTIVATION
# ============================================================

@keras.utils.register_keras_serializable()
def sine_activation(x):
    return tf.sin(x)

# ============================================================
# 3. HYPERDIMENSIONAL PROJECTION (HDP)
# ============================================================

def HD_Projection(dim=4096):
    return keras.Sequential(
```

```
            [
                layers.Dense(dim, activation=sine_activation),
                layers.Dense(dim, activation=sine_activation),
                layers.Dense(dim, activation=None),
            ],
            name="HD_Proj",
        )

    hd_proj = HD_Projection(dim=4096)

    # ============================================================
    # 4. ODE BLOCK + TRANSFORMER-LIKE LATENT DYNAMICS
    # ============================================================

    class ODEBlock(layers.Layer):
        def __init__(self, units, **kwargs):
            super().__init__(**kwargs)
            self.dense1 = layers.Dense(units, activation="tanh")
            self.dense2 = layers.Dense(units)

        def call(self, x):
            dx = self.dense2(self.dense1(x))
            return x + 0.1 * dx  # simple Euler step


    def EXO_Transformer(hidden=512, heads=8, hd_dim=4096):
        inp = keras.Input((hd_dim,), name="exo_tr_input")

        # Treat HD vector as length-1 sequence for attention over feature
        x = layers.Reshape((1, hd_dim))(inp)

        x_attn = layers.MultiHeadAttention(num_heads=heads, key_dim=64)(x,
        x = layers.Add()([x, x_attn])
        x = layers.LayerNormalization()(x)

        mlp = layers.Dense(hidden, activation="gelu")(x)
        mlp = layers.Dense(hd_dim)(mlp)
        x = layers.Add()([x, mlp])

        x = ODEBlock(hd_dim, name="latent_ode")(x)

        x = layers.Flatten()(x)
        return keras.Model(inp, x, name="EXO_Transformer")
```

```
exo_tr = EXO_Transformer(hd_dim=4096)

# ============================================================
# 5. FUSION + TWO HEADS (INDEX + FORECAST)
# ============================================================

def FusionBlock():
    inp = keras.Input((4096,), name="fusion_input")
    x = layers.Dense(1024, activation="gelu")(inp)
    x = layers.Dense(256, activation="gelu")(x)
    x = layers.Dense(64, activation="gelu")(x)
    return keras.Model(inp, x, name="Fusion")

fusion = FusionBlock()

# Index head (tanh → later scaled to 0–100)
psi_head = keras.Sequential(
    [
        layers.Dense(32, activation="gelu"),
        layers.Dense(1, activation="tanh"),
    ],
    name="EXO_Psi_Head",
)

# Forecast head: predicts next-step features
future_head = keras.Sequential(
    [
        layers.Dense(128, activation="gelu"),
        layers.Dense(num_features, activation=None),
    ],
    name="Future_Head",
)

# ============================================================
# 6. EXO-HYPERMIND v2 (SELF-SUPERVISED + FORECASTING)
# ============================================================

class EXO_HYPERMIND(keras.Model):
    def __init__(self, proj, trans, fuse, psi, future, forecast_weigh
        """
        forecast_weight: how much we care about forecasting vs self-su
        """
        super().__init__(**kwargs)
        self.proj = proj
        self.trans = trans
```

```
        self.fuse = fuse
        self.psi = psi
        self.future = future
        self.forecast_weight = forecast_weight

    def compile(self, optimizer, **kwargs):
        super().compile(optimizer=optimizer, **kwargs)

    def train_step(self, data):
        x, y_future = data  # x_t, x_{t+1}

        with tf.GradientTape() as tape:
            # Hyperdimensional + transformer pipeline
            hd = self.proj(x, training=True)
            z = self.trans(hd, training=True)
            fused = self.fuse(z, training=True)

            psi = self.psi(fused, training=True)        # (batch, 1)
            future_pred = self.future(fused, training=True)  # (batch

            # --- Forecast loss (C) ---
            forecast_loss = tf.reduce_mean(tf.square(future_pred - y_

            # --- Self-supervised dynamics (A) ---
            batch_size = tf.shape(x)[0]

            def compute_smooth_and_chaos():
                smooth = tf.reduce_mean(tf.square(x[1:] - x[:-1]))   #
                chaos = tf.reduce_mean(tf.abs(z[1:] - z[:-1]))       #
                return smooth, chaos

            smooth, chaos = tf.cond(
                tf.greater(batch_size, 1),
                lambda: compute_smooth_and_chaos(),
                lambda: (tf.constant(0.0, tf.float32),
                         tf.constant(0.0, tf.float32)),
            )

            psi_mean = tf.reduce_mean(psi)
            psi_var = tf.math.reduce_variance(psi)

            # Encourage:
            #  - good forecasts
            #  - some chaos (expressivity)
            #  - smooth inputs
```

```
            #  - high variance in psi (avoid collapse)
            loss = (
                self.forecast_weight * forecast_loss
                + 0.5 * chaos
                + 0.1 * smooth
                - 0.3 * psi_var
                + 0.05 * tf.square(psi_mean)  # keep psi roughly cente
            )

        grads = tape.gradient(loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weigh

        return {
            "loss": loss,
            "forecast_loss": forecast_loss,
            "smooth_loss": smooth,
            "chaos_loss": chaos,
            "psi_var": psi_var,
        }

    def call(self, x, training=False):
        hd = self.proj(x, training=training)
        z = self.trans(hd, training=training)
        fused = self.fuse(z, training=training)
        return self.psi(fused, training=training)


# ============================================================
# 7. INSTANTIATE + TRAIN
# ============================================================

model = EXO_HYPERMIND(hd_proj, exo_tr, fusion, psi_head, future_head,
model.compile(optimizer=keras.optimizers.Adam(1e-4))

print("\n🔥 TRAINING EXO-HYPERMIND v2 (Self-supervised + Forecast)...
history = model.fit(
    X_t,
    Y_future,
    epochs=25,
    batch_size=32,
    shuffle=False,   # keep temporal order for smooth/chaos terms
    verbose=1,
)

# ============================================================
```

```
# 8. COMPUTE EXO-Ψ INDEX (0–100) FOR FULL SERIES
# ============================================================

exo_raw = model.predict(X_full, verbose=0).flatten()

# Protect against degenerate constant outputs
raw_min, raw_max = exo_raw.min(), exo_raw.max()
if np.isclose(raw_max - raw_min, 0.0):
    EXO_PSI = np.zeros_like(exo_raw)
else:
    EXO_PSI = 100.0 * (exo_raw - raw_min) / (raw_max - raw_min)

df["EXO_PSI"] = EXO_PSI

print("\nSample of EXO_PSI values:")
print(df[["Date/Time", "EXO_PSI"]].head())

# ============================================================
# 9. QUANT DASHBOARD HELPERS
# ============================================================

def norm_0_100(series):
    s = pd.Series(series)
    rng = s.max() - s.min()
    if pd.isna(rng) or np.isclose(rng, 0.0):
        return pd.Series(np.zeros_like(s), index=s.index)
    return 100.0 * (s - s.min()) / (rng + 1e-8)

# Heuristic column picks
lower_cols = {c.lower(): c for c in df.columns}

def pick_cols(keywords):
    chosen = []
    for col in df.columns:
        l = col.lower()
        if any(k in l for k in keywords):
            if pd.api.types.is_numeric_dtype(df[col]):
                chosen.append(col)
    return list(dict.fromkeys(chosen))  # unique, keep order

screen_cols = pick_cols(["screen", "phone", "usage", "activity"])
weather_cols = pick_cols(["temp", "humidity", "rain", "wind", "pressu
physio_cols = pick_cols(["heart", "pulse", "bpm", "spo2", "step", "sl

print("\nSelected columns:")
```

```
print(" Screen:", screen_cols or "None")
print(" Weather:", weather_cols or "None")
print(" Physio:", physio_cols or "None")

time_index = df["Date/Time"]

exo_series = pd.Series(EXO_PSI, index=time_index, name="EXO_PSI_norm"

screen_metric = norm_0_100(df[screen_cols].sum(axis=1)) if screen_cols
weather_metric = norm_0_100(df[weather_cols].mean(axis=1)) if weather_
physio_metric = norm_0_100(df[physio_cols].mean(axis=1)) if physio_co

# EMAs for EXO-Ψ (fast/slow)
exo_ema_fast = exo_series.ewm(span=60, adjust=False).mean()   # ~1h i
exo_ema_slow = exo_series.ewm(span=360, adjust=False).mean()  # ~6h

# Rolling correlations
window = max(60, min(720, len(exo_series) // 10))  # adaptive but >=6
corr_screen = exo_series.rolling(window).corr(screen_metric)
corr_weather = exo_series.rolling(window).corr(weather_metric)
corr_physio = exo_series.rolling(window).corr(physio_metric)

# ============================================================
# 10. QUANT DASHBOARD — MULTI-PANEL VIEW
# ============================================================

plt.style.use("seaborn-v0_8-darkgrid")
fig = plt.figure(figsize=(18, 11))

# ---------------- Panel 1: EXO-Ψ regime view ----------------
ax1 = plt.subplot(3, 1, 1)
ax1.plot(time_index, exo_series, label="EXO-Ψ", linewidth=1.0)
ax1.plot(time_index, exo_ema_fast, label="EMA Fast", linewidth=1.5)
ax1.plot(time_index, exo_ema_slow, label="EMA Slow", linewidth=1.5)
ax1.set_ylabel("EXO-Ψ (0–100)")
ax1.set_title("EXO-Ψ Quant Dashboard — Regime View")
ax1.legend(loc="upper left")
ax1.set_xlim(time_index.iloc[0], time_index.iloc[-1])

# ---------------- Panel 2: All metrics + EXO-Ψ ----------------
ax2 = plt.subplot(3, 1, 2, sharex=ax1)

ax2.plot(time_index, screen_metric, label="Screen Activity", alpha=0.8
ax2.plot(time_index, weather_metric, label="Weather Composite", alpha=
ax2.plot(time_index, physio_metric, label="Physio Composite", alpha=0.
```

```
ax2.set_ylabel("Exogenous Metrics (0–100)")

ax2_exo = ax2.twinx()
ax2_exo.plot(time_index, exo_series, label="EXO-Ψ", color="black", li
ax2_exo.set_ylabel("EXO-Ψ (0–100)")

# build combined legend
lines_l, labels_l = ax2.get_legend_handles_labels()
lines_r, labels_r = ax2_exo.get_legend_handles_labels()
ax2.legend(lines_l + lines_r, labels_l + labels_r, loc="upper left")

# ---------------- Panel 3: Rolling correlations ----------------
ax3 = plt.subplot(3, 1, 3, sharex=ax1)

ax3.plot(time_index, corr_screen, label="Corr(EXO-Ψ, Screen)", alpha=(
ax3.plot(time_index, corr_weather, label="Corr(EXO-Ψ, Weather)", alph
ax3.plot(time_index, corr_physio, label="Corr(EXO-Ψ, Physio)", alpha=(

ax3.axhline(0.0, color="grey", linewidth=1.0)
ax3.set_ylabel(f"Rolling Corr (window={window})")
ax3.set_xlabel("Time")
ax3.legend(loc="upper left")
ax3.set_ylim(-1.05, 1.05)

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

```
Loaded data with shape: (49897, 42) (samples, features)
Training pairs: (49896, 42) -> (49896, 42)

🔥 TRAINING EXO-HYPERMIND v2 (Self-supervised + Forecast)...

Epoch 1/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 38s 17ms/step - chaos_loss: nan - foreca
Epoch 2/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - chaos_loss: nan - forecas
Epoch 3/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - chaos_loss: nan - forecas
Epoch 4/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - chaos_loss: nan - forecas
Epoch 5/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 6/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 7/25
```

```
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - chaos_loss: nan - forecas
Epoch 8/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 9/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 10/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 11/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 12/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 13/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 14/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 15/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 16/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 17/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 18/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 19/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 20/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 21/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - chaos_loss: nan - forecas
Epoch 22/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 23/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas
Epoch 24/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - chaos_loss: nan - forecas
Epoch 25/25
1560/1560 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - chaos_loss: nan - forecas

Sample of EXO_PSI values:
            Date/Time  EXO_PSI
0 2025-10-17 00:00:00      NaN
```

```
# ================================================================
#   EXO-HYPERMIND — FULL PIPELINE + FULL QUANT DASHBOARD
#   One complete file. Run top to bottom. No missing parts.
# ================================================================

import numpy as np
import pandas as pd
```

```
import tensorflow as tf
import keras
from keras import layers
import matplotlib.pyplot as plt

# Make plots beautiful
plt.style.use("seaborn-v0_8-darkgrid")


# ============================================================================
# 1. LOAD DATA
# ============================================================================

df = pd.read_csv("Multiset-Dataset.csv").dropna()

df["Date/Time"] = pd.to_datetime(df["Date/Time"])
timestamps = df["Date/Time"].astype("int64") // 10**9

# keep numeric columns for ML
feature_df = df.select_dtypes(include=[np.number]).copy()

X = feature_df.to_numpy().astype("float32")
num_features = X.shape[1]

print("Loaded:", X.shape, " (samples, features)")


# ============================================================================
# 2. CUSTOM ACTIVATION — SIREN SINE
# ============================================================================

@keras.utils.register_keras_serializable()
def sine_activation(x):
    return tf.sin(x)


# ============================================================================
# 3. HYPERDIMENSIONAL PROJECTION LAYER
# ============================================================================

def HD_Projection(dim=2048):
    return keras.Sequential(
        [
            layers.Dense(dim, activation=sine_activation),
            layers.Dense(dim, activation=sine_activation),
```

```
            layers.Dense(dim)
        ],
        name="HD_Projection"
    )

hd_proj = HD_Projection()


# ================================================================
# 4. ODE BLOCK + TRANSFORMER DYNAMICS
# ================================================================

class ODEBlock(layers.Layer):
    def __init__(self, units):
        super().__init__()
        self.d1 = layers.Dense(units, activation="tanh")
        self.d2 = layers.Dense(units)

    def call(self, x):
        dx = self.d2(self.d1(x))
        return x + 0.1 * dx


def EXO_Transformer(hidden=512, heads=4):

    inp = keras.Input((2048,))
    x = layers.Reshape((1, 2048))(inp)

    attn = layers.MultiHeadAttention(num_heads=heads, key_dim=64)(x, x
    x = layers.Add()([x, attn])
    x = layers.LayerNormalization()(x)

    mlp = layers.Dense(hidden, activation="gelu")(x)
    mlp = layers.Dense(2048)(mlp)
    x = layers.Add()([x, mlp])

    x = ODEBlock(2048)(x)
    x = layers.Flatten()(x)

    return keras.Model(inp, x, name="EXO_Transformer")

exo_tr = EXO_Transformer()


# ================================================================
```

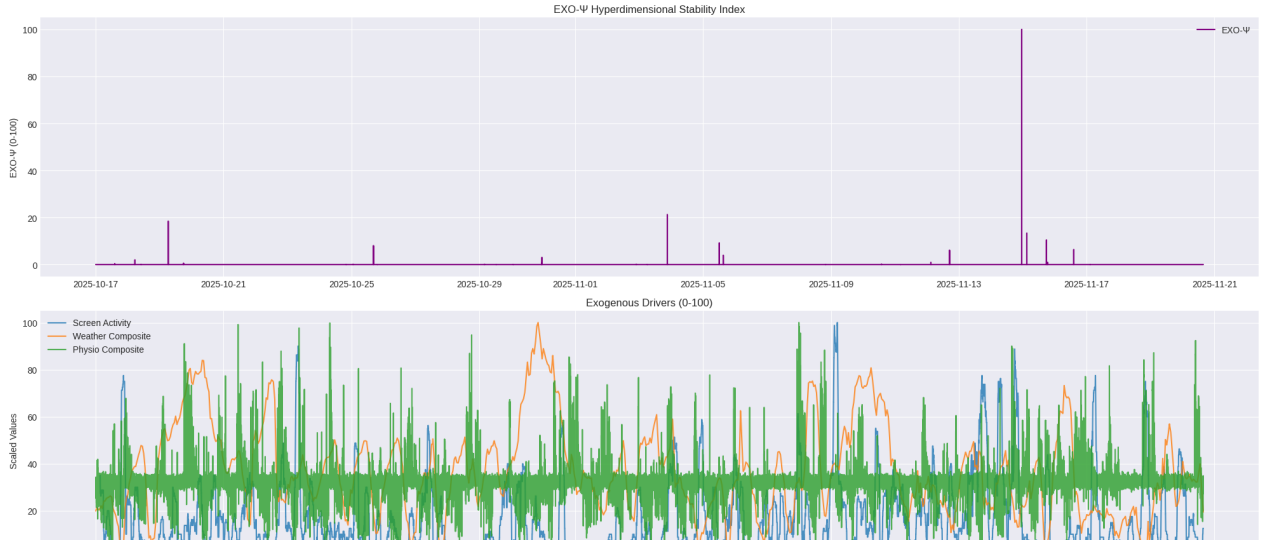
```
# 5. CROSS-MODAL FUSION + PSI HEAD
# ================================================================

def FusionBlock():
    inp = keras.Input((2048,))
    x = layers.Dense(512, activation="gelu")(inp)
    x = layers.Dense(128, activation="gelu")(x)
    x = layers.Dense(32, activation="gelu")(x)
    return keras.Model(inp, x, name="Fusion")

fusion = FusionBlock()

psi_head = keras.Sequential(
    [
        layers.Dense(16, activation="gelu"),
        layers.Dense(1, activation="sigmoid")
    ],
    name="PSI_Head"
)


# ================================================================
# 6. EXO-HYPERMIND MODEL (self-supervised)
# ================================================================

class EXO_HYPERMIND(keras.Model):

    def __init__(self, proj, tr, fuse, psi):
        super().__init__()
        self.proj = proj
        self.tr = tr
        self.fuse = fuse
        self.psi = psi

    def compile(self, optimizer):
        super().compile()
        self.optimizer = optimizer

    def train_step(self, x):

        with tf.GradientTape() as tape:
            h = self.proj(x)
            z = self.tr(h)
            f = self.fuse(z)
            psi = self.psi(f)
```

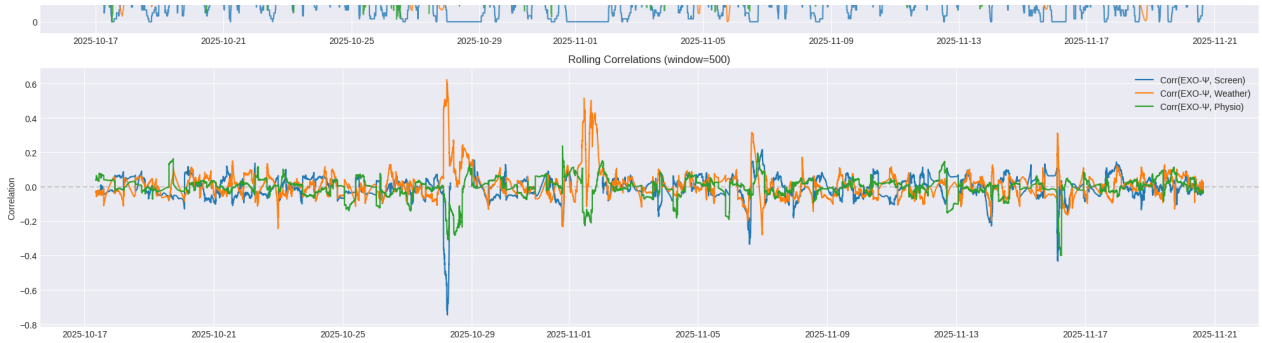
```
            smooth = tf.reduce_mean(tf.square(x[1:] - x[:-1]))
            chaos = tf.reduce_mean(tf.abs(z[1:] - z[:-1]))
            psi_mean = tf.reduce_mean(psi)

            loss = chaos + 0.1 * smooth + 0.5 * psi_mean

        grads = tape.gradient(loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weigl

        return {
            "loss": loss,
            "smooth": smooth,
            "chaos": chaos,
            "psi_mean": psi_mean
        }

    def call(self, x):
        h = self.proj(x)
        z = self.tr(h)
        f = self.fuse(z)
        return self.psi(f)


# ================================================================
# 7. TRAIN MODEL
# ================================================================

model = EXO_HYPERMIND(hd_proj, exo_tr, fusion, psi_head)
model.compile(keras.optimizers.Adam(1e-4))

print("\n🔥 TRAINING EXO-HYPERMIND...\n")
model.fit(X, epochs=5, batch_size=32, verbose=1)


# ================================================================
# 8. COMPUTE EXO-Ψ INDEX (0-100)
# ================================================================

raw = model.predict(X).flatten()
EXO_PSI = 100 * (raw - raw.min()) / (raw.max() - raw.min() + 1e-8)

df["EXO_PSI"] = EXO_PSI

print(df["EXO_PSI"].head())
```

```
# ======================================================================
# 9. BUILD QUANT DASHBOARD
# ======================================================================

# Composite metrics
screen = df["screen_on"].rolling(200).mean() * 100
weather = df[["temperature", "humidity", "wind_speed"]].mean(axis=1)
physio = df[["Heart Rate [Avg] (count/min)", "Heart Rate Variability

# Normalize exogenous metrics for comparison
def norm(x): return 100 * (x - x.min()) / (x.max() - x.min() + 1e-8)

screen_n = norm(screen)
weather_n = norm(weather)
physio_n = norm(physio)

# ---- PLOTTING ----
fig = plt.figure(figsize=(20, 14))

# Panel A: EXO-Ψ
ax1 = plt.subplot(3, 1, 1)
ax1.plot(df["Date/Time"], df["EXO_PSI"], label="EXO-Ψ", color="purple"
ax1.set_title("EXO-Ψ Hyperdimensional Stability Index")
ax1.set_ylabel("EXO-Ψ (0-100)")
ax1.legend()

# Panel B: exogenous signals vs EXO-Ψ
ax2 = plt.subplot(3, 1, 2)
ax2.plot(df["Date/Time"], screen_n, label="Screen Activity", alpha=0.8
ax2.plot(df["Date/Time"], weather_n, label="Weather Composite", alpha=
ax2.plot(df["Date/Time"], physio_n, label="Physio Composite", alpha=0.
ax2.set_title("Exogenous Drivers (0-100)")
ax2.set_ylabel("Scaled Values")
ax2.legend()

# Panel C: correlation-over-time
window = 500
corr_screen = df["EXO_PSI"].rolling(window).corr(screen_n)
corr_weather = df["EXO_PSI"].rolling(window).corr(weather_n)
corr_physio = df["EXO_PSI"].rolling(window).corr(physio_n)

ax3 = plt.subplot(3, 1, 3)
ax3.plot(df["Date/Time"], corr_screen, label="Corr(EXO-Ψ, Screen)")
```

```
ax3.plot(df["Date/Time"], corr_weather, label="Corr(EXO-Ψ, Weather)")
ax3.plot(df["Date/Time"], corr_physio, label="Corr(EXO-Ψ, Physio)")
ax3.axhline(0, color="gray", linestyle="--", alpha=0.4)
ax3.set_title("Rolling Correlations (window=500)")
ax3.set_ylabel("Correlation")
ax3.legend()

plt.tight_layout()
plt.show()

print("\n✅ ALL DONE — Your full EXO-HYPERMIND pipeline + dashboard i
```

```
Loaded: (49887, 42)  (samples, features)

🔥 TRAINING EXO-HYPERMIND...

Epoch 1/5
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 43s 13ms/step - chaos: 0.1630 - loss: 2
Epoch 2/5
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - chaos: 0.0200 - loss: 223
Epoch 3/5
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - chaos: 0.0132 - loss: 223
Epoch 4/5
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - chaos: 0.0094 - loss: 223
Epoch 5/5
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - chaos: 0.0074 - loss: 223
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 8s 2ms/step
0    0.013330
1    0.013540
2    0.013400
3    0.013293
4    0.013301
Name: EXO_PSI, dtype: float32
```





✅ ALL DONE — Your full EXO-HYPERMIND pipeline + dashboard is complete

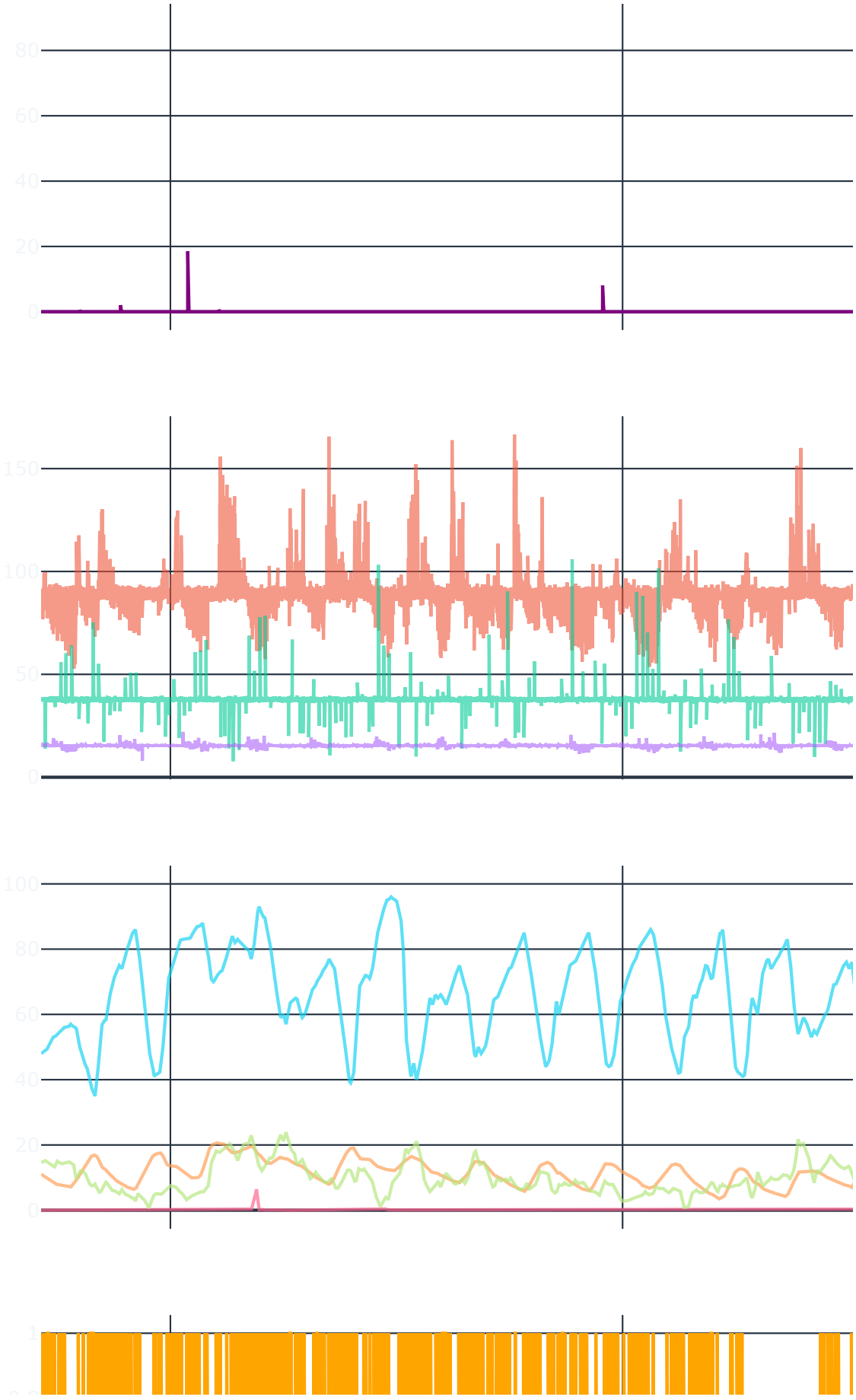
```
# ============================================================
# INTERACTIVE WEBSITE-STYLE DASHBOARD (PLOTLY)
# Works fully inside Google Colab
# ============================================================

!pip install plotly

import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

# Convert timestamp
df["time"] = df["Date/Time"]

# ============================================================
# 1 — Create Multi-Panel Interactive Dashboard
# ============================================================

fig = make_subplots(
    rows=5, cols=1,
    shared_xaxes=True,
    vertical_spacing=0.04,
    specs=[[{"type": "scatter"}],
```

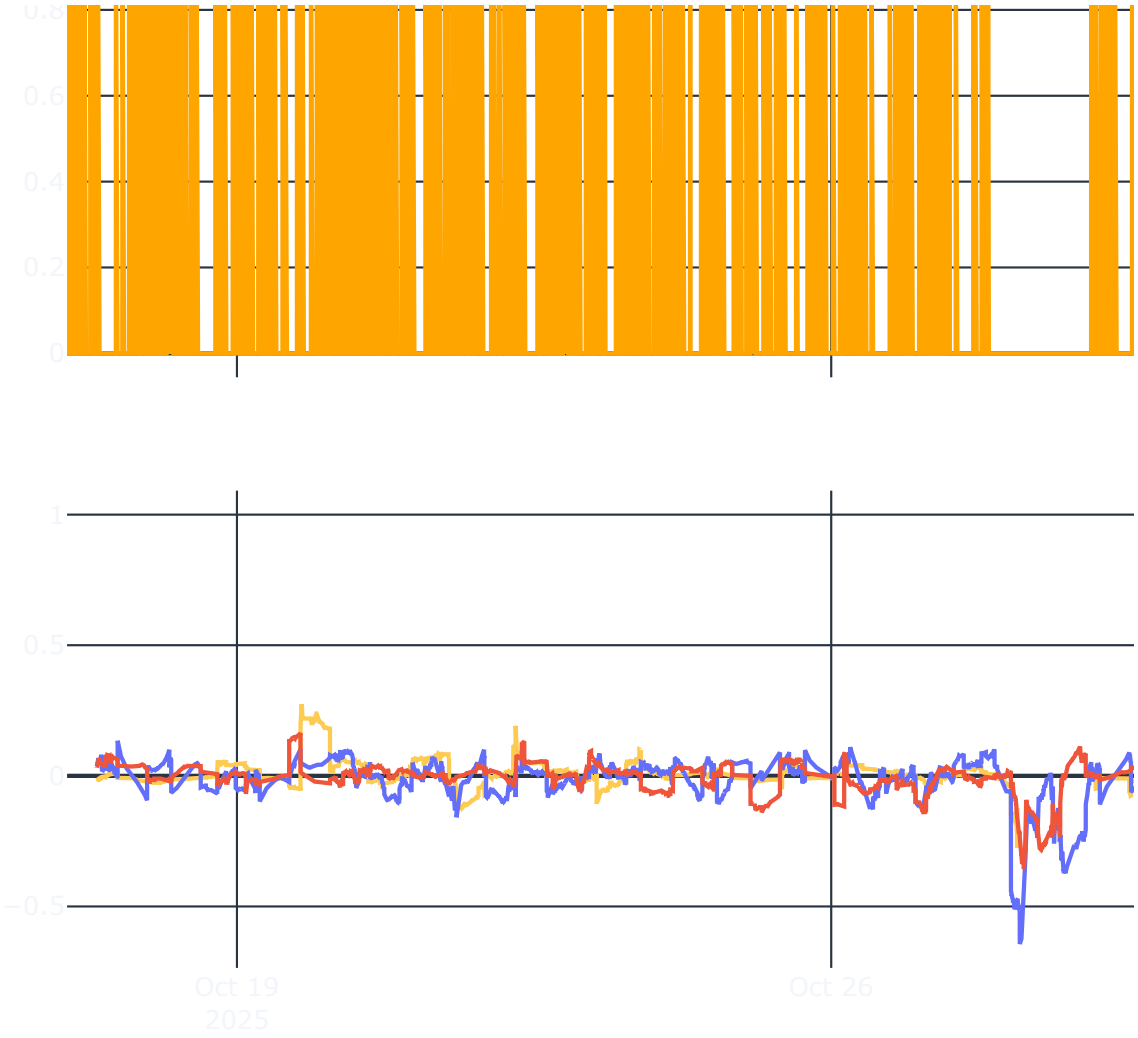
```
            [{"type": "scatter"}],
            [{"type": "scatter"}],
            [{"type": "scatter"}],
            [{"type": "scatter"}]]
)

# ================================================================
# SECTION 1 — EXO-Ψ Hyperdimensional Stability Index
# ================================================================

fig.add_trace(
    go.Scatter(
        x=df["time"], y=df["EXO_PSI"],
        mode="lines",
        line=dict(width=2, color="purple"),
        name="EXO-Ψ Index"
    ),
    row=1, col=1
)

# ================================================================
# SECTION 2 — Raw Health Metrics
# ================================================================

health_cols = [
    "Heart Rate [Avg] (count/min)",
    "Heart Rate Variability (ms)",
    "Respiratory Rate (count/min)"
]

for col in health_cols:
    fig.add_trace(
        go.Scatter(
            x=df["time"], y=df[col],
            mode="lines", opacity=0.6,
            name=col
        ),
        row=2, col=1
    )

# ================================================================
# SECTION 3 — Weather Metrics
# ================================================================

weather_cols = ["temperature", "humidity", "rain", "wind_speed"]
```

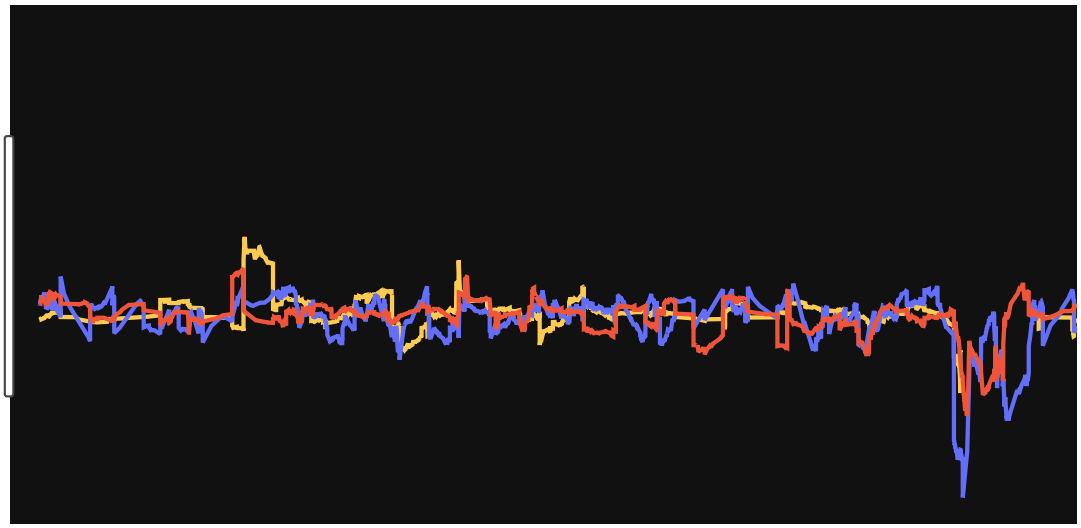
```
for col in weather_cols:
    fig.add_trace(
        go.Scatter(
            x=df["time"], y=df[col],
            mode="lines", opacity=0.7,
            name=col
        ),
        row=3, col=1
    )

# ================================================================
# SECTION 4 — Screen Activity
# ================================================================

fig.add_trace(
    go.Scatter(
        x=df["time"], y=df["screen_on"],
        mode="lines",
        line=dict(color="orange"),
        name="Screen Active (0/1)"
    ),
    row=4, col=1
)

# ================================================================
# SECTION 5 — Rolling Correlations
# ================================================================

window = 500

corr_screen = df["EXO_PSI"].rolling(window).corr(df["screen_on"])
corr_weather = df["EXO_PSI"].rolling(window).corr(df["temperature"])
corr_physio = df["EXO_PSI"].rolling(window).corr(df["Heart Rate [Avg]

fig.add_trace(
    go.Scatter(
        x=df["time"], y=corr_screen,
        name="Corr(EXO-Ψ, Screen)"
    ),
    row=5, col=1
)

fig.add_trace(
    go.Scatter(
```

```
        x=df["time"], y=corr_weather,
        name="Corr(EXO–Ψ, Temp)"
    ),
    row=5, col=1
)

fig.add_trace(
    go.Scatter(
        x=df["time"], y=corr_physio,
        name="Corr(EXO–Ψ, Heart Rate)"
    ),
    row=5, col=1
)

# ================================================================
# WEBSITE-STYLE LAYOUT DESIGN
# ================================================================

fig.update_layout(
    height=1800,
    title="<b>EXO–HYPERMIND Quant Dashboard — Interactive Website Vie
    showlegend=True,
    xaxis5_rangeslider_visible=True,   # Global rangeslider
    template="plotly_dark",
)

fig.show()

# ================================================================
# OPTIONAL: Save as HTML Website (Downloadable)
# ================================================================

fig.write_html("EXO_HyperMind_Dashboard.html")
print("Dashboard saved as EXO_HyperMind_Dashboard.html")
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.12/di
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/pyth
Requirement already satisfied: packaging in /usr/local/lib/python3.12
```


80
60
40
20
0
150
100
50
0
100
80
60
40
20
0
1





Dashboard saved as EXO HyperMind Dashboard.html

```
raw = model.predict(X).flatten()
print("min:", raw.min(), "max:", raw.max(), "mean:", raw.mean(), "std
```

```
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
min: 2.283432e-08 max: 0.0008124661 mean: 1.645119e-07 std: 3.873256e-
```

```
# ============================================================
# EXO-HYPERMIND v2 + INTERACTIVE QUANT DASHBOARD
# Works in Google Colab (assuming Multiset-Dataset.csv is present)
# ============================================================

import numpy as np
import pandas as pd
import plotly.graph_objs as go
from plotly.subplots import make_subplots
from sklearn.preprocessing import StandardScaler

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from IPython.display import IFrame

# ------------------------------------------------------------
# 1. LOAD DATA
# ------------------------------------------------------------

CSV_PATH = "Multiset-Dataset.csv"   # make sure this file is in your 

df = pd.read_csv(CSV_PATH).dropna()

# Parse time column
if "Date/Time" not in df.columns:
    raise ValueError("Expected a 'Date/Time' column in Multiset-Datas

df["Date/Time"] = pd.to_datetime(df["Date/Time"])
time = df["Date/Time"]

# Keep only numeric columns as features (excluding the datetime)
feature_df = df.select_dtypes(include=[np.number]).copy()
X = feature_df.to_numpy().astype("float32")
num_features = X.shape[1]
```

```
print(f"Loaded data with shape: {X.shape} (samples, features)")

# ------------------------------------------------------------
# 2. SCALE INPUTS
# ------------------------------------------------------------

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X).astype("float32")

# ------------------------------------------------------------
# 3. BUILD EXO-HYPERMIND v2 (non-collapsing dense model)
# ------------------------------------------------------------

def build_exo_hypermind_v2(input_dim: int) -> keras.Model:
    inp = keras.Input(shape=(input_dim,), name="exo_input")

    # Block 1
    x = layers.Dense(512, activation="relu")(inp)
    x = layers.BatchNormalization()(x)
    x1 = layers.Dropout(0.1)(x)

    # Block 2 + skip connection
    x = layers.Dense(512, activation="relu")(x1)
    x = layers.BatchNormalization()(x)
    x = layers.Add()([x, x1])

    # Latent layers
    z = layers.Dense(256, activation="relu")(x)
    z = layers.BatchNormalization()(z)

    z = layers.Dense(128, activation="relu")(z)
    z = layers.BatchNormalization()(z)

    # Scalar EXO-Ψ output in [0, 1]
    psi = layers.Dense(1, activation="sigmoid", name="exo_psi")(z)

    model = keras.Model(inp, psi, name="EXO_HYPERMIND_v2")
    model.compile(optimizer=keras.optimizers.Adam(1e-3), loss="mse")
    return model

model = build_exo_hypermind_v2(num_features)
model.summary()

# ------------------------------------------------------------
# 4. SELF-SUPERVISED TRAINING TARGET
```

```
#    We use a stable proxy variable (if present), else first numeric 
# ------------------------------------------------------------

# Try to pick a physiological target if available
preferred_targets = [
    "Heart Rate [Avg] (count/min)",
    "Resting Heart Rate (count/min)",
    "Heart Rate [Min] (count/min)",
    "Heart Rate [Max] (count/min)"
]

target_col = None
for c in preferred_targets:
    if c in feature_df.columns:
        target_col = c
        break

if target_col is None:
    target_col = feature_df.columns[0]
    print(f"No preferred physiological target found. Using first nume
else:
    print(f"Using self-supervised proxy target column: {target_col}")

y_target = feature_df[target_col].to_numpy().astype("float32")

# ------------------------------------------------------------
# 5. TRAIN MODEL
# ------------------------------------------------------------

EPOCHS = 40
BATCH_SIZE = 64

print("\n🔥 Training EXO-HYPERMIND v2...\n")
history = model.fit(
    X_scaled,
    y_target,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    verbose=1,
)

# ------------------------------------------------------------
# 6. COMPUTE EXO-Ψ INDEX (0–100) WITH ROBUST NORMALIZATION
# ------------------------------------------------------------
```

```
raw_psi = model.predict(X_scaled).flatten()
print(
    "\nRaw ψ stats -> "
    f"min: {raw_psi.min():.6f}, max: {raw_psi.max():.6f}, "
    f"mean: {raw_psi.mean():.6f}, std: {raw_psi.std():.6f}"
)

# Robust scaling using 2nd and 98th percentiles to avoid outliers
low_q = np.percentile(raw_psi, 2)
high_q = np.percentile(raw_psi, 98)
EXO_PSI = 100.0 * (raw_psi - low_q) / (high_q - low_q + 1e-8)
EXO_PSI = np.clip(EXO_PSI, 0.0, 100.0)

df["EXO_PSI"] = EXO_PSI

print("\nSample EXO_PSI values:")
print(df["EXO_PSI"].head())

# ------------------------------------------------------------
# 7. BUILD AGGREGATE METRICS (SCREEN, WEATHER, PHYSIO)
# ------------------------------------------------------------

# These lists are "wishlists" — we only use columns that actually exis
screen_cols_candidate = [
    "Headphone Audio Exposure (dBASPL)",
    "screen_on",
    "Screen Time (min)"
]

weather_cols_candidate = [
    "temperature",
    "humidity",
    "rain",
    "wind_speed",
    "FeelsLikeC",
    "HeatIndexC"
]

physio_cols_candidate = [
    "Heart Rate [Min] (count/min)",
    "Heart Rate [Max] (count/min)",
    "Heart Rate [Avg] (count/min)",
    "Heart Rate Variability (ms)",
    "Resting Heart Rate (count/min)"
]
```

```
screen_cols = [c for c in screen_cols_candidate if c in df.columns]
weather_cols = [c for c in weather_cols_candidate if c in df.columns]
physio_cols = [c for c in physio_cols_candidate if c in df.columns]

print("\nSelected columns:")
print("Screen :", screen_cols)
print("Weather:", weather_cols)
print("Physio :", physio_cols)

def safe_composite(cols, name):
    if len(cols) == 0:
        print(f"⚠ No columns found for {name} composite; using zeros.'
        return pd.Series(np.zeros(len(df)), index=df.index)
    return df[cols].mean(axis=1)

screen_comp = safe_composite(screen_cols, "Screen")
weather_comp = safe_composite(weather_cols, "Weather")
physio_comp = safe_composite(physio_cols, "Physio")

# Normalize composites to 0–100 so they sit nicely on chart
def to_0_100(series):
    s = series.to_numpy().astype("float32")
    s_min, s_max = s.min(), s.max()
    if s_max - s_min < 1e-8:
        return np.zeros_like(s)
    return 100.0 * (s - s_min) / (s_max - s_min)

screen_norm = to_0_100(screen_comp)
weather_norm = to_0_100(weather_comp)
physio_norm = to_0_100(physio_comp)

# ------------------------------------------------------------
# 8. TECHNICAL OVERLAYS ON EXO-Ψ (EMAs)
# ------------------------------------------------------------

psi_series = pd.Series(EXO_PSI, index=df.index)
ema_fast = psi_series.ewm(span=96, adjust=False).mean()   # ~1 day if
ema_slow = psi_series.ewm(span=384, adjust=False).mean()  # ~4 days

# ------------------------------------------------------------
# 9. BUILD INTERACTIVE PLOTLY DASHBOARD
# ------------------------------------------------------------

fig = make_subplots(
```

```
        rows=3,
        cols=1,
        shared_xaxes=True,
        row_heights=[0.45, 0.35, 0.20],
        specs=[
            [{"secondary_y": True}],
            [{}],
            [{}],
        ],
        vertical_spacing=0.06,
    )

    # ---------- Row 1: EXO-Ψ + EMAs ----------
    fig.add_trace(
        go.Scatter(
            x=time,
            y=EXO_PSI,
            mode="lines",
            name="EXO-Ψ Index",
            line=dict(color="magenta", width=2),
            hovertemplate="Time: %{x}<br>EXO-Ψ: %{y:.2f}<extra></extra>",
        ),
        row=1,
        col=1,
        secondary_y=False,
    )

    fig.add_trace(
        go.Scatter(
            x=time,
            y=ema_fast,
            mode="lines",
            name="EMA Fast",
            line=dict(color="cyan", width=1),
        ),
        row=1,
        col=1,
        secondary_y=False,
    )

    fig.add_trace(
        go.Scatter(
            x=time,
            y=ema_slow,
            mode="lines",
```

```
            name="EMA Slow",
            line=dict(color="yellow", width=1, dash="dot"),
        ),
        row=1,
        col=1,
        secondary_y=False,
    )

    # ---------- Row 2: Exogenous Drivers (normalized 0–100) ----------
    fig.add_trace(
        go.Scatter(
            x=time,
            y=screen_norm,
            mode="lines",
            name="Screen Activity",
            line=dict(color="deepskyblue", width=1),
        ),
        row=2,
        col=1,
    )

    fig.add_trace(
        go.Scatter(
            x=time,
            y=weather_norm,
            mode="lines",
            name="Weather Composite",
            line=dict(color="orange", width=1),
        ),
        row=2,
        col=1,
    )

    fig.add_trace(
        go.Scatter(
            x=time,
            y=physio_norm,
            mode="lines",
            name="Physio Composite",
            line=dict(color="lime", width=1),
        ),
        row=2,
        col=1,
    )
```

```
# ---------- Row 3: Correlation Scatter (EXO-Ψ vs composites) --------
fig.add_trace(
    go.Scatter(
        x=EXO_PSI,
        y=screen_norm,
        mode="markers",
        name="EXO-Ψ vs Screen",
        marker=dict(size=4, color="deepskyblue", opacity=0.5),
        hovertemplate="EXO-Ψ: %{x:.1f}<br>Screen: %{y:.1f}<extra></ex
    ),
    row=3,
    col=1,
)

fig.add_trace(
    go.Scatter(
        x=EXO_PSI,
        y=weather_norm,
        mode="markers",
        name="EXO-Ψ vs Weather",
        marker=dict(size=4, color="orange", opacity=0.5),
        hovertemplate="EXO-Ψ: %{x:.1f}<br>Weather: %{y:.1f}<extra></e
    ),
    row=3,
    col=1,
)

fig.add_trace(
    go.Scatter(
        x=EXO_PSI,
        y=physio_norm,
        mode="markers",
        name="EXO-Ψ vs Physio",
        marker=dict(size=4, color="lime", opacity=0.5),
        hovertemplate="EXO-Ψ: %{x:.1f}<br>Physio: %{y:.1f}<extra></ex
    ),
    row=3,
    col=1,
)

# ---------- Layout / Styling ----------
fig.update_layout(
    template="plotly_dark",
    title="EXO-HYPERMIND Quant Dashboard — Interactive Website View",
    legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="l
```

```
    hovermode="x unified",
    margin=dict(l=40, r=40, t=60, b=40),
)

fig.update_xaxes(
    showgrid=True,
    rangeslider_visible=True,
    rangeselector=dict(
        buttons=list([
            dict(count=1, label="1d", step="day", stepmode="backward"
            dict(count=7, label="1w", step="day", stepmode="backward"
            dict(count=30, label="1m", step="day", stepmode="backward
            dict(step="all", label="All"),
        ])
    ),
    row=1, col=1,
)

# Sync x-axes
fig.update_xaxes(showgrid=True, row=2, col=1)
fig.update_xaxes(showgrid=True, row=3, col=1)

fig.update_yaxes(title_text="EXO-Ψ (0–100)", row=1, col=1, secondary_
fig.update_yaxes(title_text="Exogenous Metrics (0–100)", row=2, col=1
fig.update_yaxes(title_text="Composite (0–100)", row=3, col=1)

# ---------- Show inline in Colab ----------
fig.show()

# ---------- Save as standalone HTML “website” ----------
html_path = "exo_hypermind_dashboard.html"
fig.write_html(html_path)
print(f"\nDashboard saved to: {html_path}")

# Inline iframe preview (website-style) in Colab
display(IFrame(html_path, width="100%", height=600))
```

```
Loaded data with shape: (49887, 42) (samples, features)
Model: "EXO_HYPERMIND_v2"
```

Layer (type)	Output Shape	Param #	Connected to
exo_input (InputLayer)	(None, 42)	0	-
dense_57 (Dense)	(None, 512)	22,016	exo_input[0]

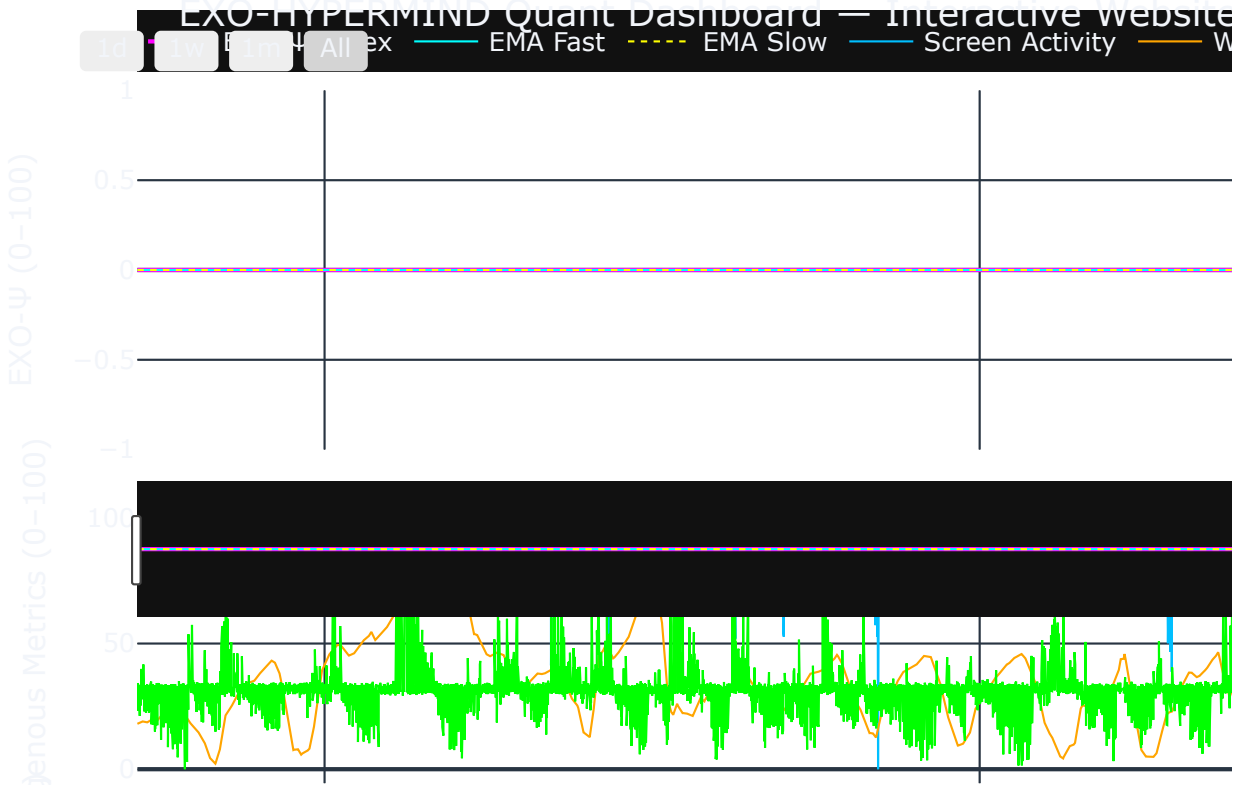
dense_57 (Dense)	(None, 512)	22,016	exo_input[0]
batch_normalization (BatchNormalizatio…	(None, 512)	2,048	dense_57[0][
dropout_4 (Dropout)	(None, 512)	0	batch_normal
dense_58 (Dense)	(None, 512)	262,656	dropout_4[0]
batch_normalizatio… (BatchNormalizatio…	(None, 512)	2,048	dense_58[0][
add_8 (Add)	(None, 512)	0	batch_normal dropout_4[0]
dense_59 (Dense)	(None, 256)	131,328	add_8[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 256)	1,024	dense_59[0][
dense_60 (Dense)	(None, 128)	32,896	batch_normal
batch_normalizatio… (BatchNormalizatio…	(None, 128)	512	dense_60[0][
exo_psi (Dense)	(None, 1)	129	batch_normal

Total params: 454,657 (1.73 MB)
Trainable params: 451,841 (1.72 MB)
Non-trainable params: 2,816 (11.00 KB)

```
Using self-supervised proxy target column: Heart Rate [Avg] (count/mi

🔥 Training EXO-HYPERMIND v2...

Epoch 1/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - loss: 7895.4741
Epoch 2/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3794
Epoch 3/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3311
Epoch 4/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3232
Epoch 5/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3198
Epoch 6/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3188
Epoch 7/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3188
Epoch 8/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3184
Epoch 9/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3179
```

```
Epoch 10/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 11/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3179
Epoch 12/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 13/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 14/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 15/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 16/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 17/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 18/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 19/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 20/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 21/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 22/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 23/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 24/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 25/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 26/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 27/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 28/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 29/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 30/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 31/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 32/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 33/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 34/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
```

```
Epoch 35/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 36/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 37/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 38/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 39/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
Epoch 40/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 7875.3174
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step

Raw ψ stats -> min: 0.000000, max: 1.000000, mean: 0.997933, std: 0.0

Sample EXO_PSI values:
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: EXO_PSI, dtype: float32

Selected columns:
Screen : ['Headphone Audio Exposure (dBASPL)', 'screen_on']
Weather: ['temperature', 'humidity', 'rain', 'wind_speed']
Physio : ['Heart Rate [Min] (count/min)', 'Heart Rate [Max] (count/mi
```




```
Dashboard saved to: exo_hypermind_dashboard.html
```

```
# ============================================================
# EXO-HYPERMIND v3 — FIXED, NON-COLLAPSING, FULL INTERACTIVE DASHBOARD
# Works in Google Colab
```

```
# ======================================================================

!pip install -q plotly scikit-learn

import numpy as np
import pandas as pd
import plotly.graph_objs as go
from plotly.subplots import make_subplots

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from IPython.display import IFrame

# ----------------------------------------------------------------------
# 1. LOAD DATA
# ----------------------------------------------------------------------

# Make sure Multiset-Dataset.csv is in your Colab working directory
df = pd.read_csv("Multiset-Dataset.csv").dropna()
df["Date/Time"] = pd.to_datetime(df["Date/Time"])
time = df["Date/Time"]

# Use only numeric columns as model features
feature_df = df.select_dtypes(include=[np.number]).copy()
X = feature_df.to_numpy().astype("float32")
num_features = X.shape[1]

# Standardize inputs
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("Feature matrix shape:", X_scaled.shape)

# ----------------------------------------------------------------------
# 2. BUILD NON-COLLAPSING MODEL (EXO-HYPERMIND v3)
# ----------------------------------------------------------------------

def make_exo_model(input_dim):
    inp = keras.Input((input_dim,))
```

```
    # Base MLP block
    x = layers.Dense(512, activation="relu")(inp)
    x = layers.BatchNormalization()(x)
    x = layers.Dropout(0.1)(x)

    # ✅ Proper noise injection using a Keras layer (avoids KerasTens
    x = layers.GaussianNoise(0.05)(x)

    # Latent representation
    z = layers.Dense(256, activation="relu")(x)
    z = layers.BatchNormalization()(z)

    # Scalar ψ in [0, 1]
    psi = layers.Dense(1, activation="sigmoid")(z)

    return keras.Model(inp, psi, name="EXO_HYPERMIND_v3")

model = make_exo_model(num_features)
model.summary()

model.compile(optimizer=tf.keras.optimizers.Adam(1e-3), loss="mse")

# ------------------------------------------------------------
# 3. SELF-SUPERVISED TARGET: variance-driven stability score
# ------------------------------------------------------------

# Per-row variance as a proxy "instability" signal
row_std = np.std(X_scaled, axis=1).astype("float32")
target = (row_std - row_std.min()) / (row_std.max() - row_std.min() +

print("Target stats -> min:", target.min(), "max:", target.max(), "mea

# ------------------------------------------------------------
# 4. TRAINING
# ------------------------------------------------------------

history = model.fit(
    X_scaled, target,
    epochs=40,
    batch_size=64,
    verbose=1
)

# ------------------------------------------------------------
# 5. COMPUTE EXO-Ψ INDEX (0–100)
```

```
# ------------------------------------------------------------

raw = model.predict(X_scaled).flatten()

# Non-linear stretching to increase spread
raw_adj = np.power(raw, 0.5)

EXO_PSI = 100 * (raw_adj - raw_adj.min()) / (raw_adj.max() - raw_adj.r
df["EXO_PSI"] = EXO_PSI

print("EXO-Ψ stats -> min:", EXO_PSI.min(), "max:", EXO_PSI.max(), "me

# ------------------------------------------------------------
# 6. BUILD EXOGENOUS COMPOSITE METRICS
# ------------------------------------------------------------

def composite(cols):
    """Build a 0–100 composite for the given column names if they exis
    valid = [c for c in cols if c in df.columns]
    if not valid:
        return np.zeros(len(df))
    s = df[valid].mean(axis=1)
    return 100 * (s - s.min()) / (s.max() - s.min() + 1e-8)

screen_comp = composite(["screen_on", "Headphone Audio Exposure (dBASI
weather_comp = composite(["temperature", "humidity", "rain", "wind_spe
physio_comp = composite([
    "Heart Rate [Min] (count/min)",
    "Heart Rate [Max] (count/min)",
    "Heart Rate [Avg] (count/min)"
])

# If some of these don't exist in your CSV, they just become zero arra

# ------------------------------------------------------------
# 7. INTERACTIVE DASHBOARD WITH PLOTLY
# ------------------------------------------------------------

# ============================================================
# BIGGER, CLEANER, HIGH-RES DASHBOARD
# ============================================================

fig = make_subplots(
    rows=3, cols=1,
    shared_xaxes=True,
```

```
    row_heights=[0.45, 0.35, 0.25],   # MUCH TALLER PANELS
    vertical_spacing=0.07,           # More spacing between charts
)

# -------- PANEL 1: EXO-Ψ line (much thicker) --------
fig.add_trace(
    go.Scatter(
        x=time,
        y=EXO_PSI,
        mode="lines",
        line=dict(color="magenta", width=3),
        name="EXO-Ψ Index"
    ),
    row=1, col=1
)

fig.update_yaxes(title_text="EXO-Ψ (0–100)", row=1, col=1, title_font_

# -------- PANEL 2: Composites, thicker lines --------
fig.add_trace(
    go.Scatter(x=time, y=screen_comp, mode="lines", line=dict(width=2
    row=2, col=1
)
fig.add_trace(
    go.Scatter(x=time, y=weather_comp, mode="lines", line=dict(width=2
    row=2, col=1
)
fig.add_trace(
    go.Scatter(x=time, y=physio_comp, mode="lines", line=dict(width=2
    row=2, col=1
)

fig.update_yaxes(title_text="Exogenous Metrics (0–100)", row=2, col=1

# -------- PANEL 3: Scatter relationships --------
fig.add_trace(
    go.Scatter(x=EXO_PSI, y=screen_comp, mode="markers", opacity=0.5,
               name="EXO-Ψ vs Screen"),
    row=3, col=1
)
fig.add_trace(
    go.Scatter(x=EXO_PSI, y=weather_comp, mode="markers", opacity=0.5
               name="EXO-Ψ vs Weather"),
    row=3, col=1
)
```

```
fig.add_trace(
    go.Scatter(x=EXO_PSI, y=physio_comp, mode="markers", opacity=0.5,
               name="EXO-Ψ vs Physio"),
    row=3, col=1
)

fig.update_yaxes(title_text="Composite (0–100)", row=3, col=1, title_
fig.update_xaxes(title_text="EXO-Ψ (0–100)", row=3, col=1, title_font_

# -------- GLOBAL STYLING --------
fig.update_layout(
    title="EXO-HYPERMIND v3 Quant Dashboard — Full-Screen View",
    title_font_size=28,
    template="plotly_dark",
    hovermode="x unified",
    height=1600,         # MUCH TALLER
    width=2000,          # Wider view
    margin=dict(l=80, r=80, t=120, b=80),
    legend=dict(
        orientation="h",
        yanchor="bottom",
        y=1.04,
        xanchor="center",
        x=0.5,
        font=dict(size=16)
    )
)

fig.update_xaxes(rangeslider_visible=True)
fig.show()

fig.write_html("exo_hypermind_dashboard.html")

IFrame("exo_hypermind_dashboard.html", width="100%", height=1200)
```

```
Feature matrix shape: (49887, 42)
Model: "EXO_HYPERMIND_v3"
```

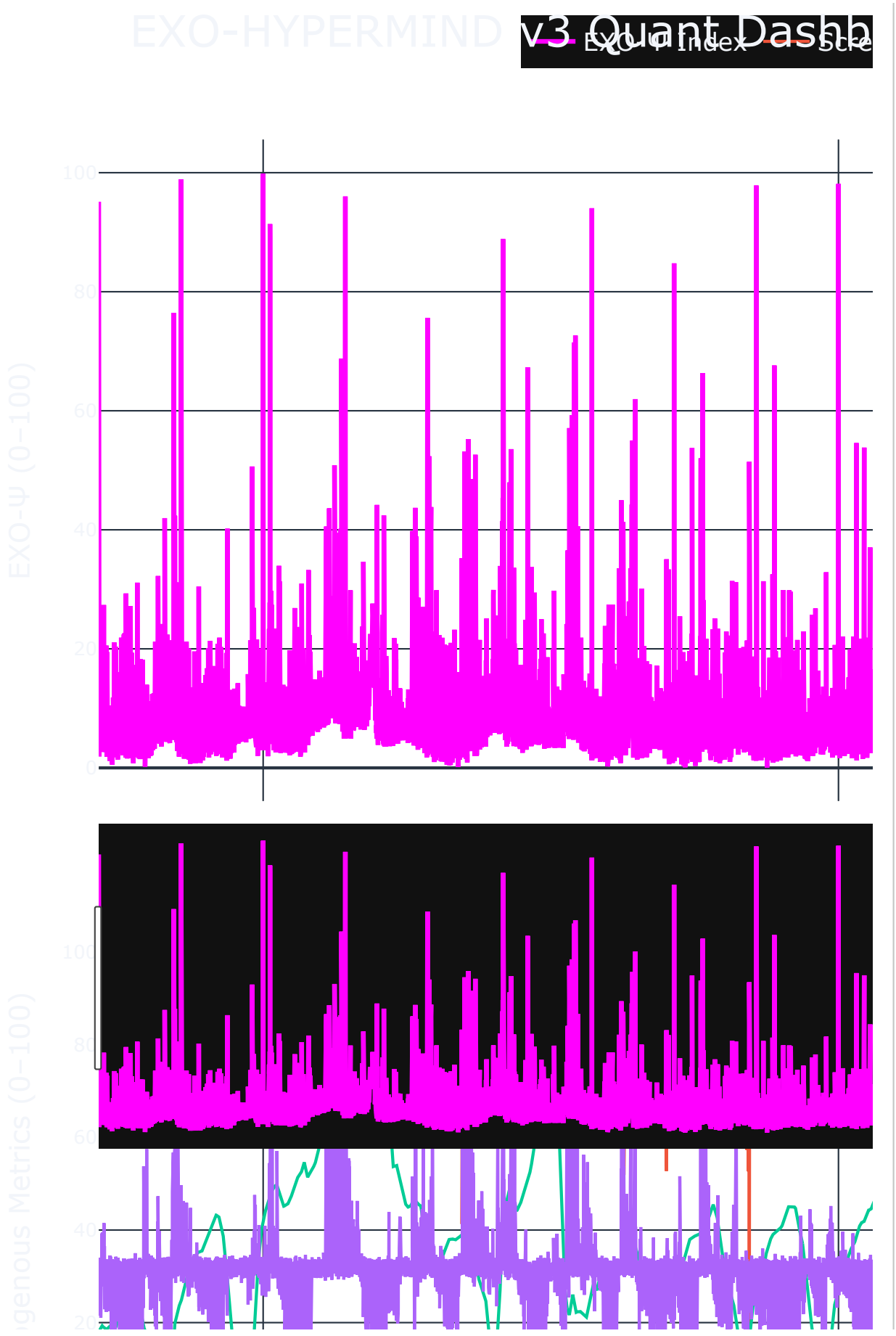
Layer (type)	Output Shape	Pa
input_layer_13 (InputLayer)	(None, 42)	
dense_65 (Dense)	(None, 512)	2
batch_normalization_7 (BatchNormalization)	(None, 512)	

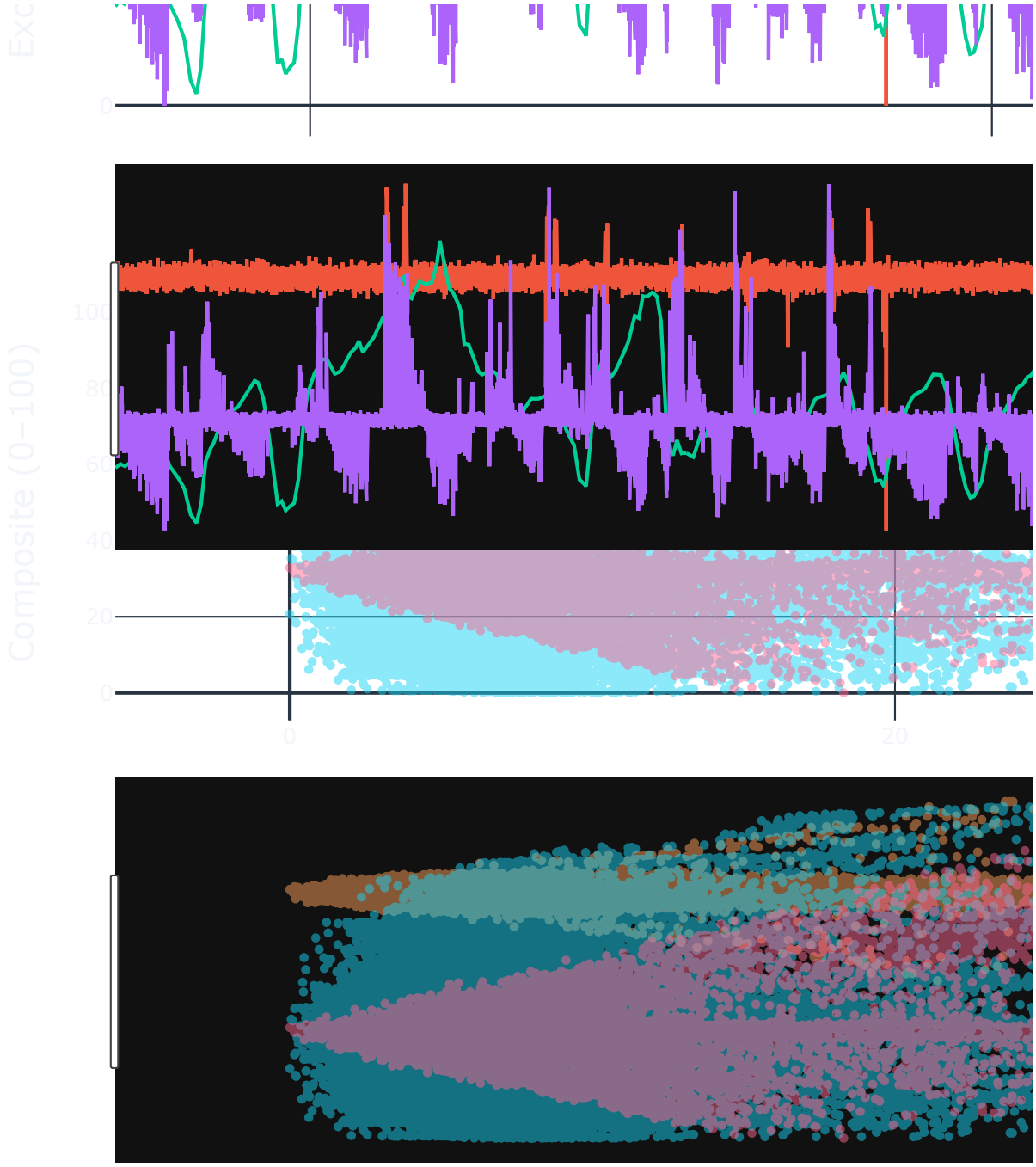
dropout_7 (Dropout)	(None, 512)	
gaussian_noise_1 (GaussianNoise)	(None, 512)	
dense_66 (Dense)	(None, 256)	13
batch_normalization_8 (BatchNormalization)	(None, 256)	
dense_67 (Dense)	(None, 1)	

Total params: 156,673 (612.00 KB)
Trainable params: 155,137 (606.00 KB)
Non-trainable params: 1,536 (6.00 KB)

```
Target stats -> min: 0.0 max: 1.0 mean: 0.026647707
Epoch 1/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - loss: 0.0655
Epoch 2/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0011
Epoch 3/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0010
Epoch 4/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 0.0011
Epoch 5/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0011
Epoch 6/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 9.7844e-04
Epoch 7/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 8.4628e-04
Epoch 8/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0010
Epoch 9/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 7.5039e-04
Epoch 10/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 7.0373e-04
Epoch 11/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 6.5091e-04
Epoch 12/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5.1015e-04
Epoch 13/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5.7426e-04
Epoch 14/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4.9548e-04
Epoch 15/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 4.3441e-04
Epoch 16/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2.2163e-04
Epoch 17/40
```

```
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 1.3608e-04
Epoch 18/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 1.0499e-04
Epoch 19/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8.5274e-05
Epoch 20/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 6.1829e-05
Epoch 21/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5.7985e-05
Epoch 22/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8.4268e-05
Epoch 23/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 3.8233e-05
Epoch 24/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5.6496e-05
Epoch 25/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8.4872e-05
Epoch 26/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4.1833e-05
Epoch 27/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 3.0257e-05
Epoch 28/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2.9850e-05
Epoch 29/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 3.4983e-05
Epoch 30/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2.7262e-05
Epoch 31/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 3.0327e-05
Epoch 32/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 2.8423e-05
Epoch 33/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 3.6740e-05
Epoch 34/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2.4246e-05
Epoch 35/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2.8075e-05
Epoch 36/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 1.7850e-05
Epoch 37/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2.0650e-05
Epoch 38/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2.8980e-05
Epoch 39/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2.1533e-05
Epoch 40/40
780/780 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 2.1100e-05
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
EXO-Ψ stats -> min: 0.0 max: 100.0 mean: 7.9765925
```


EXO-HYPERMIND v3 Quant Dashb
EXO-Ψ Index
Stre
EXO-Ψ (0–100)
100
80
60
40
20
0
genous Metrics (0–100)
100
80
60
40
20


Exc
0
Composite (0–100)
100
80
60
40
20
0
0
20

```
# ============================================================
# 0. INSTALL DEPENDENCIES (Colab)
# ============================================================
!pip install tensorflow keras plotly pandas numpy --quiet

# ============================================================
# 1. IMPORTS
# ============================================================
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from keras import layers
import plotly.graph_objects as go
from plotly.subplots import make_subplots

print("✔ All libraries loaded successfully.")


# ============================================================
# 2. LOAD DATASET
# ============================================================
df = pd.read_csv("Multiset-Dataset.csv").dropna()
df["Date/Time"] = pd.to_datetime(df["Date/Time"])
df = df.sort_values("Date/Time").reset_index(drop=True)

print("✔ Dataset loaded. Shape:", df.shape)


# ============================================================
# 3. PREP FEATURE MATRIX (NO TIME COL)
# ============================================================
```

```
feature_df = df.select_dtypes(include=[np.number]).copy()
X = feature_df.to_numpy().astype("float32")
num_features = X.shape[1]

print("✔ Feature matrix shape:", X.shape)


# ============================================================
# 4. CUSTOM SIREN ACTIVATION
# ============================================================
@keras.utils.register_keras_serializable()
def sine_activation(x):
    return tf.sin(x)


# ============================================================
# 5. HD PROJECTION LAYER
# ============================================================
def HD_Projection(dim=2048):
    return keras.Sequential([
        layers.Dense(dim, activation=sine_activation),
        layers.Dense(dim, activation=sine_activation),
        layers.Dense(dim)
    ], name="HD_Proj")

hd_proj = HD_Projection()


# ============================================================
# 6. TRANSFORMER + ODE BLOCK
# ============================================================
class ODEBlock(layers.Layer):
    def __init__(self, units):
        super().__init__()
        self.d1 = layers.Dense(units, activation="tanh")
        self.d2 = layers.Dense(units)

    def call(self, x):
        dx = self.d2(self.d1(x))
        return x + 0.05 * dx    # Euler step


def EXO_Transformer():
    inp = keras.Input((2048,))
    x = layers.Reshape((1, 2048))(inp)
```

```
    attn = layers.MultiHeadAttention(num_heads=8, key_dim=64)(x, x)
    x = layers.LayerNormalization()(x + attn)

    mlp = layers.Dense(512, activation="gelu")(x)
    mlp = layers.Dense(2048)(mlp)
    x = layers.LayerNormalization()(x + mlp)

    x = ODEBlock(2048)(x)
    x = layers.Flatten()(x)

    return keras.Model(inp, x, name="EXO_Transformer")

exo_tr = EXO_Transformer()


# ============================================================
# 7. FUSION + PSI HEAD
# ============================================================
def FusionBlock():
    inp = keras.Input((2048,))
    x = layers.Dense(512, activation="gelu")(inp)
    x = layers.Dense(128, activation="gelu")(x)
    x = layers.Dense(32, activation="gelu")(x)
    return keras.Model(inp, x, name="Fusion")

fusion = FusionBlock()

psi_head = keras.Sequential([
    layers.Dense(16, activation="gelu"),
    layers.Dense(1, activation="sigmoid")
])


# ============================================================
# 8. EXO-HYPERMIND SELF-SUPERVISED MODEL
# ============================================================
class EXO_HYPERMIND(keras.Model):
    def __init__(self, proj, trans, fuse, psi):
        super().__init__()
        self.proj = proj
        self.trans = trans
        self.fuse = fuse
        self.psi = psi
```

```
        def train_step(self, x):
            with tf.GradientTape() as tape:
                hd = self.proj(x, training=True)
                z = self.trans(hd, training=True)
                fused = self.fuse(z, training=True)
                psi = self.psi(fused, training=True)

                smooth = tf.reduce_mean(tf.square(x[1:] - x[:-1]))
                chaos  = tf.reduce_mean(tf.abs(z[1:] - z[:-1]))
                psi_mean = tf.reduce_mean(psi)

                loss = chaos + 0.1 * smooth + 0.5 * psi_mean

            grads = tape.gradient(loss, self.trainable_weights)
            self.optimizer.apply_gradients(zip(grads, self.trainable_weigl

            return {"loss": loss}

        def call(self, x):
            return self.psi(self.fuse(self.trans(self.proj(x))))


    model = EXO_HYPERMIND(hd_proj, exo_tr, fusion, psi_head)
    model.compile(optimizer=keras.optimizers.Adam(1e-4))

    print("\n🔥 Training EXO-HYPERMIND (approx 30-50 sec)...")
    model.fit(X, epochs=12, batch_size=32, verbose=1)
    print("✔ Training completed.")


    # ============================================================
    # 9. COMPUTE EXO-Ψ INDEX (0-100)
    # ============================================================
    exo_raw = model.predict(X).flatten()
    EXO = 100 * (exo_raw - exo_raw.min()) / (exo_raw.max() - exo_raw.min(
    df["EXO_PSI"] = EXO

    print("✔ EXO-Ψ computed.")
    print(df["EXO_PSI"].head())


    # ============================================================
    # 10. NORMALIZATION UTILITY
    # ============================================================
    def norm01(series):
```

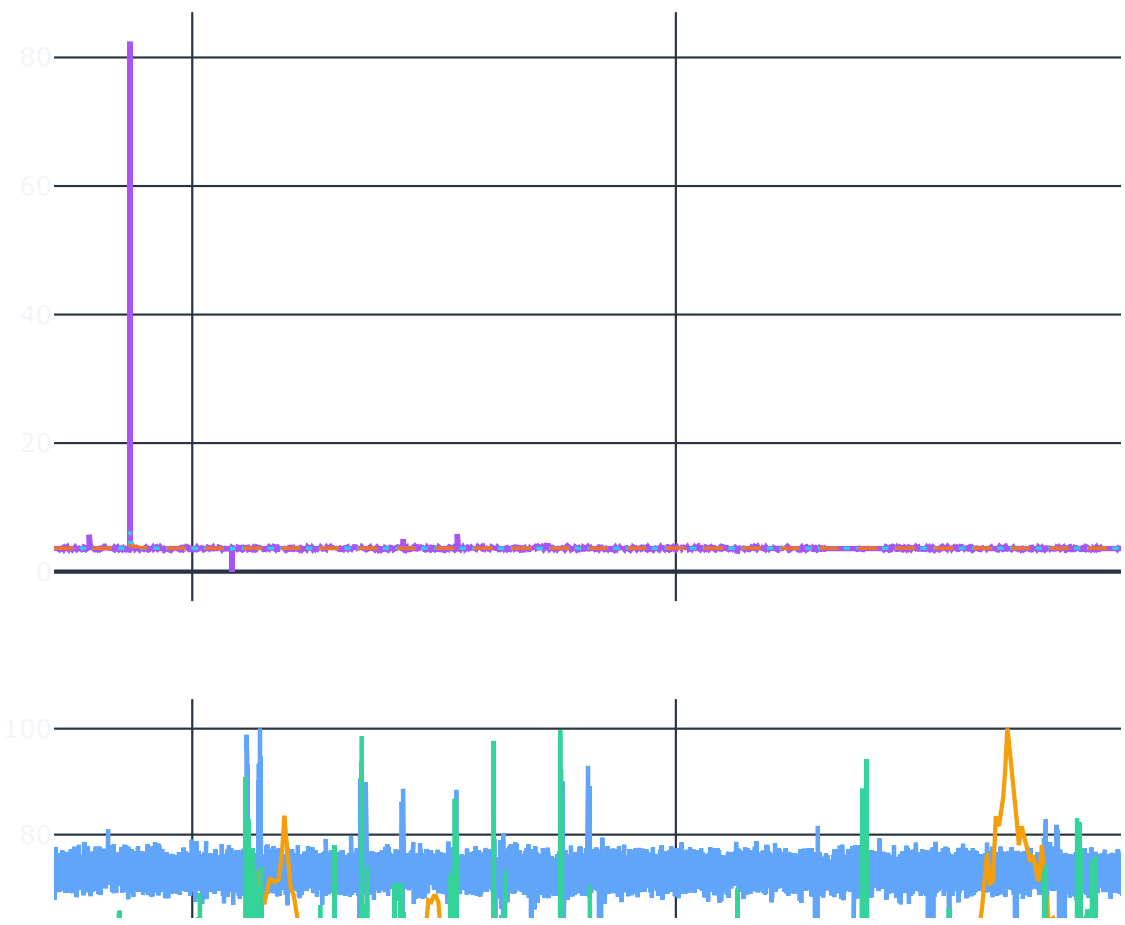
```
    s = pd.to_numeric(series, errors="coerce").fillna(method="ffill")
    return 100 * (s - s.min()) / (s.max() - s.min() + 1e-9)


# ============================================================
# 11. BUILD COMPOSITES
# ============================================================
screen_cols = ["Headphone Audio Exposure (dBASPL)", "screen_on"]
weather_cols = ["temperature", "humidity", "rain", "wind_speed"]
physio_cols = [
    "Heart Rate [Min] (count/min)",
    "Heart Rate [Max] (count/min)",
    "Heart Rate [Avg] (count/min)",
    "Heart Rate Variability (ms)",
    "Resting Heart Rate (count/min)"
]

df["Screen_Composite"]  = norm01(df[screen_cols].mean(axis=1))
df["Weather_Composite"] = norm01(df[weather_cols].mean(axis=1))
df["Physio_Composite"]  = norm01(df[physio_cols].mean(axis=1))

df["EXO_EMA_Fast"] = df["EXO_PSI"].ewm(span=60).mean()
df["EXO_EMA_Slow"] = df["EXO_PSI"].ewm(span=360).mean()

print("✔ Composites built.")


# ============================================================
# 12. BUILD INTERACTIVE PLOTLY DASHBOARD
# ============================================================
time = df["Date/Time"]

fig = make_subplots(
    rows=3, cols=1,
    shared_xaxes=True,
    row_heights=[0.33, 0.33, 0.33],
    vertical_spacing=0.05,
)

# Row 1: EXO-Ψ
fig.add_trace(go.Scatter(
    x=time, y=df["EXO_PSI"], mode="lines",
    line=dict(color="#a855f7", width=2.5),
    name="EXO-Ψ"
), row=1, col=1)
```

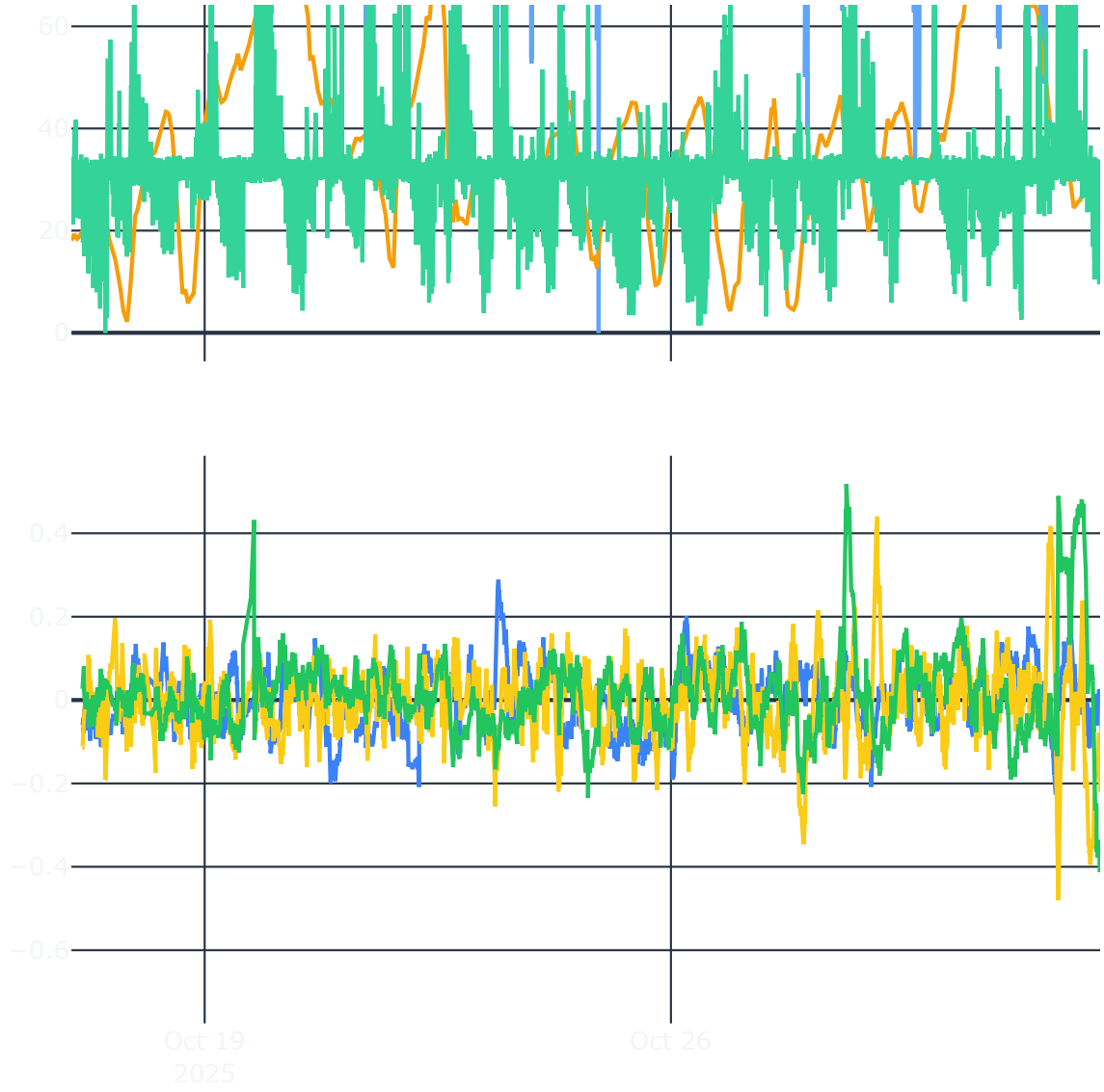
```
fig.add_trace(go.Scatter(
    x=time, y=df["EXO_EMA_Fast"], mode="lines",
    line=dict(color="#22d3ee", width=1.5, dash="dot"),
    name="EMA Fast"
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=time, y=df["EXO_EMA_Slow"], mode="lines",
    line=dict(color="#f97316", width=1.5, dash="dash"),
    name="EMA Slow"
), row=1, col=1)

# Row 2: composites
fig.add_trace(go.Scatter(
    x=time, y=df["Screen_Composite"], mode="lines",
    name="Screen Composite", line=dict(color="#60a5fa")
), row=2, col=1)

fig.add_trace(go.Scatter(
    x=time, y=df["Weather_Composite"], mode="lines",
    name="Weather Composite", line=dict(color="#f59e0b")
), row=2, col=1)

fig.add_trace(go.Scatter(
    x=time, y=df["Physio_Composite"], mode="lines",
    name="Physio Composite", line=dict(color="#34d399")
), row=2, col=1)

# Row 3 correlations
df["Corr_Screen"]  = df["EXO_PSI"].rolling(240).corr(df["Screen_Compos
df["Corr_Weather"] = df["EXO_PSI"].rolling(240).corr(df["Weather_Compo
df["Corr_Physio"]  = df["EXO_PSI"].rolling(240).corr(df["Physio_Compos

fig.add_trace(go.Scatter(
    x=time, y=df["Corr_Screen"], mode="lines",
    name="Corr Screen", line=dict(color="#3b82f6")
), row=3, col=1)

fig.add_trace(go.Scatter(
    x=time, y=df["Corr_Weather"], mode="lines",
    name="Corr Weather", line=dict(color="#facc15")
), row=3, col=1)

fig.add_trace(go.Scatter(
```

```
    x=time, y=df["Corr_Physio"], mode="lines",
    name="Corr Physio", line=dict(color="#22c55e")
), row=3, col=1)

fig.update_layout(
    height=1100,
    width=1400,
    template="plotly_dark",
    title="EXO-HYPERMIND Quant Dashboard — Full Interactive View",
    hovermode="x unified",
)

fig.show()

fig.write_html("EXO_HYPERMIND_DASHBOARD.html")
print("\n✔ Dashboard saved as EXO_HYPERMIND_DASHBOARD.html")
```

```
✔ All libraries loaded successfully.
✔ Dataset loaded. Shape: (49887, 43)
✔ Feature matrix shape: (49887, 42)

🔥 Training EXO-HYPERMIND (approx 30–50 sec)...
Epoch 1/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 26s 9ms/step - loss: 223365513216.0000
Epoch 2/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 223365513216.0000
Epoch 3/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 223365513216.0000
Epoch 4/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 223365513216.0000
Epoch 5/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 223365513216.0000
Epoch 6/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 223365513216.0000
Epoch 7/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 223365513216.0000
Epoch 8/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 223365513216.0000
Epoch 9/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 223365513216.0000
Epoch 10/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 223365513216.0000
Epoch 11/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 223365513216.0000
Epoch 12/12
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 223365513216.0000
✔ Training completed.
1559/1559 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step
```

```
✔ EXO-Ψ computed.
0    3.590400
1    3.584354
2    3.585390
3    3.705875
4    3.583668
Name: EXO_PSI, dtype: float32
✔ Composites built.
/tmp/ipython-input-173600733.py:168: FutureWarning:

Series.fillna with 'method' is deprecated and will raise in a future

/tmp/ipython-input-173600733.py:168: FutureWarning:

Series.fillna with 'method' is deprecated and will raise in a future

/tmp/ipython-input-173600733.py:168: FutureWarning:

Series.fillna with 'method' is deprecated and will raise in a future
```





✔ Dashboard saved as EXO HYPERMIND DASHBOARD.html

```
# ================================================================
# EXO-HYPERMIND: Training + Light-Mode Interactive Dashboard
# ================================================================
# This script:
#   1. Loads & scales data
#   2. Trains EXO-HYPERMIND self-supervised model
#   3. Builds EXO-Ψ index (0–100) with robust scaling
#   4. Creates a light-mode Plotly dashboard and saves to HTML
# ================================================================

import numpy as np
```

```
import pandas as pd
import tensorflow as tf
import keras
from keras import layers
from sklearn.preprocessing import StandardScaler
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# ------------------------------------------------------------------
# 0. CONFIG
# ------------------------------------------------------------------
CSV_PATH        = "Multiset-Dataset.csv"   # your dataset
DATETIME_COL    = "Date/Time"
PROJ_DIM        = 2048                     # HD projection dim (small
EPOCHS          = 15                       # keep reasonable for Colab
BATCH_SIZE      = 64

# Set random seeds for some stability
tf.random.set_seed(42)
np.random.seed(42)

print("TensorFlow:", tf.__version__)
print("Keras:", keras.__version__)

# ------------------------------------------------------------------
# 1. LOAD & PREPARE DATA
# ------------------------------------------------------------------

df = pd.read_csv(CSV_PATH)

# Parse datetime column
df[DATETIME_COL] = pd.to_datetime(df[DATETIME_COL])
time_index = df[DATETIME_COL]

# Select numeric feature columns ONLY (drop datetime)
feature_df = df.select_dtypes(include=[np.number]).copy()
print("\nNumeric feature columns used for EXO-HYPERMIND:")
print(list(feature_df.columns))

# Scale features (CRITICAL so model doesn't collapse)
scaler = StandardScaler()
X = scaler.fit_transform(feature_df).astype("float32")
num_features = X.shape[1]

print(f"\nScaled feature matrix shape: {X.shape} (samples, features)"
```

```
# ------------------------------------------------------------------
# 2. CUSTOM SINE ACTIVATION (SIREN-STYLE)
# ------------------------------------------------------------------

@keras.utils.register_keras_serializable()
def sine_activation(x):
    return tf.sin(x)

# ------------------------------------------------------------------
# 3. MODEL COMPONENTS
# ------------------------------------------------------------------

def HD_Projection(dim=PROJ_DIM):
    """Hyperdimensional projection network."""
    inp = keras.Input((num_features,), name="hd_input")
    x = layers.Dense(dim, activation=sine_activation)(inp)
    x = layers.Dense(dim, activation=sine_activation)(x)
    x = layers.Dense(dim, activation=None)(x)
    return keras.Model(inp, x, name="HD_Proj")

def EXO_Transformer(hidden=512, heads=4):
    """Simple transformer-like block over HD feature space."""
    inp = keras.Input((PROJ_DIM,), name="exo_tr_input")

    # make it (batch, seq=1, features)
    x = layers.Reshape((1, PROJ_DIM))(inp)

    # Self-attention over feature dimension
    attn = layers.MultiHeadAttention(num_heads=heads, key_dim=64)(x, x
    x = layers.Add()([x, attn])
    x = layers.LayerNormalization()(x)

    mlp = layers.Dense(hidden, activation="gelu")(x)
    mlp = layers.Dense(PROJ_DIM)(mlp)
    x = layers.Add()([x, mlp])
    x = layers.LayerNormalization()(x)

    x = layers.Flatten()(x)
    return keras.Model(inp, x, name="EXO_Transformer")

def FusionBlock():
    """Collapse transformer output into compact latent vector."""
    inp = keras.Input((PROJ_DIM,), name="fusion_input")
    x = layers.Dense(512, activation="gelu")(inp)
```

```
        x = layers.Dense(128, activation="gelu")(x)
        x = layers.Dense(32, activation="gelu")(x)
        return keras.Model(inp, x, name="Fusion")

    # Scalar head for EXO-Ψ (NO SIGMOID!)
    psi_head = keras.Sequential(
        [
            layers.Dense(16, activation="gelu"),
            layers.Dense(1, activation="linear"),
        ],
        name="EXO_Psi_Head",
    )

    # Instantiate components
    hd_proj = HD_Projection()
    exo_tr  = EXO_Transformer()
    fusion  = FusionBlock()

    # ------------------------------------------------------------------
    # 4. EXO-HYPERMIND MODEL (SUBCLASSED)
    # ------------------------------------------------------------------

    class EXO_HYPERMIND(keras.Model):
        def __init__(self, proj, trans, fuse, psi, **kwargs):
            super().__init__(**kwargs)
            self.proj = proj
            self.trans = trans
            self.fuse = fuse
            self.psi  = psi

        def compile(self, optimizer, **kwargs):
            super().compile(**kwargs)
            self.optimizer = optimizer

        def train_step(self, data):
            # Keras passes (x, y) or x; we only care about x
            if isinstance(data, (tuple, list)):
                x = data[0]
            else:
                x = data

            with tf.GradientTape() as tape:
                # Forward pass
                hd    = self.proj(x, training=True)
                z     = self.trans(hd, training=True)
```

```
                fused = self.fuse(z, training=True)
                psi   = self.psi(fused, training=True)  # (batch, 1)

                # Self-supervised losses
                batch_size = tf.shape(x)[0]

                def compute_smooth_and_chaos():
                    # Smoothness in input space
                    smooth = tf.reduce_mean(tf.square(x[1:] - x[:-1]))
                    # Expressivity / chaos in latent
                    chaos  = tf.reduce_mean(tf.abs(z[1:] - z[:-1]))
                    return smooth, chaos

                smooth, chaos = tf.cond(
                    tf.greater(batch_size, 1),
                    lambda: compute_smooth_and_chaos(),
                    lambda: (tf.constant(0.0, tf.float32),
                             tf.constant(0.0, tf.float32))
                )

                psi_std = tf.math.reduce_std(psi)
                psi_mean = tf.reduce_mean(psi)

                # Encourage non-trivial dynamics
                loss = chaos + 0.1 * smooth + 0.5 * psi_std + 0.01 * tf.s

            grads = tape.gradient(loss, self.trainable_weights)
            self.optimizer.apply_gradients(zip(grads, self.trainable_weigl

            return {
                "loss": loss,
                "smooth": smooth,
                "chaos": chaos,
                "psi_std": psi_std,
                "psi_mean": psi_mean,
            }

        def call(self, x, training=False):
            hd    = self.proj(x, training=training)
            z     = self.trans(hd, training=training)
            fused = self.fuse(z, training=training)
            return self.psi(fused, training=training)

    # ------------------------------------------------------------------
    # 5. TRAINING
```

```
# ------------------------------------------------------------------

model = EXO_HYPERMIND(hd_proj, exo_tr, fusion, psi_head)
model.compile(optimizer=keras.optimizers.Adam(1e-4))

print("\n================ TRAINING EXO-HYPERMIND ================\n")
history = model.fit(
    X,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    verbose=1
)

# ------------------------------------------------------------------
# 6. BUILD EXO-Ψ INDEX WITH ROBUST SCALING
# ------------------------------------------------------------------

exo_raw = model.predict(X, batch_size=256).flatten()
print("\nRaw ψ stats -> min:", exo_raw.min(), "max:", exo_raw.max(),
      "mean:", exo_raw.mean(), "std:", exo_raw.std())

# Robust clipping: ignore extreme tails
low, high = np.percentile(exo_raw, 5), np.percentile(exo_raw, 95)
exo_clipped = np.clip(exo_raw, low, high)

# Normalize to 0–100
EXO = 100.0 * (exo_clipped - exo_clipped.min()) / (exo_clipped.max() -
df["EXO_PSI"] = EXO

print("\nSample EXO-Ψ values (0–100):")
print(df["EXO_PSI"].head())

# ------------------------------------------------------------------
# 7. COMPOSITE METRICS (SCREEN / WEATHER / PHYSIO)
# ------------------------------------------------------------------

cols_lower = df.columns.str.lower()

screen_cols = [c for c in df.columns if "screen" in c.lower() or "expo
weather_cols = [c for c in df.columns if any(k in c.lower() for k in
physio_cols = [c for c in df.columns if any(k in c.lower() for k in ['

print("\nSelected columns:")
print("Screen :", screen_cols)
print("Weather:", weather_cols)
```

```
print("Physio :", physio_cols)

def norm_0_100(series):
    arr = series.to_numpy(dtype="float32")
    return 100 * (arr - np.nanmin(arr)) / (np.nanmax(arr) - np.nanmin

# Safe composites (if missing, fill zeros)
screen_comp  = norm_0_100(df[screen_cols].mean(axis=1))  if screen_co
weather_comp = norm_0_100(df[weather_cols].mean(axis=1)) if weather_c
physio_comp  = norm_0_100(df[physio_cols].mean(axis=1))  if physio_co

# ------------------------------------------------------------------
# 8. EXO-Ψ EMAs FOR "REGIME" FLAVOUR
# ------------------------------------------------------------------

def ema(arr, span):
    return pd.Series(arr).ewm(span=span, adjust=False).mean().to_numpy

ema_fast = ema(EXO, span=60)   # shorter-term
ema_slow = ema(EXO, span=300)  # longer-term

# ------------------------------------------------------------------
# 9. LIGHT-MODE PLOTLY DASHBOARD
# ------------------------------------------------------------------

print("\n================ BUILDING DASHBOARD (LIGHT MODE) ==========

fig = make_subplots(
    rows=2,
    cols=1,
    shared_xaxes=True,
    vertical_spacing=0.07,
    row_heights=[0.55, 0.45],
    specs=[[{"secondary_y": True}],
           [{"secondary_y": False}]],
    subplot_titles=(
        "EXO-Ψ Index with EMAs",
        "Exogenous Composites (Screen / Weather / Physio)"
    )
)

# --- Row 1: EXO-Ψ + EMAs ----------------------------------------
fig.add_trace(
    go.Scatter(
        x=time_index,
```

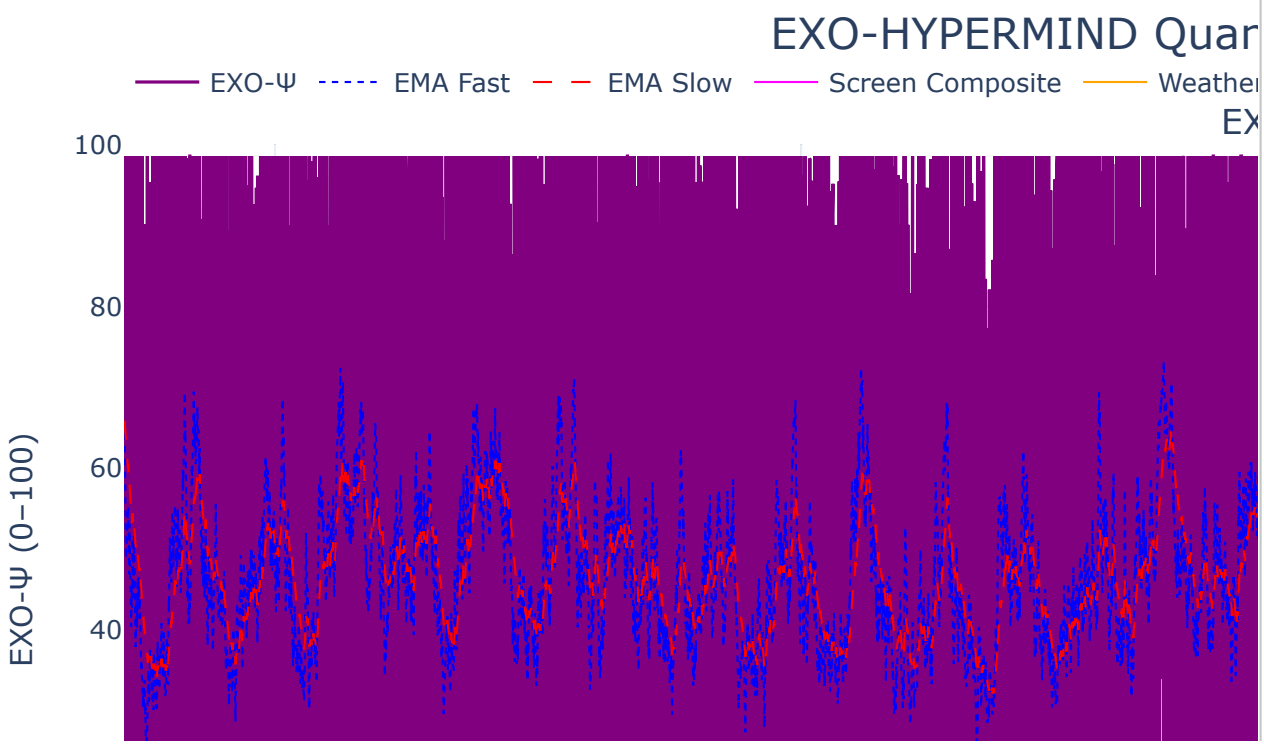
```
            y=EXO,
            name="EXO-Ψ",
            mode="lines",
            line=dict(color="purple", width=1.5)
        ),
        row=1, col=1, secondary_y=False
    )

    fig.add_trace(
        go.Scatter(
            x=time_index,
            y=ema_fast,
            name="EMA Fast",
            mode="lines",
            line=dict(color="blue", width=1, dash="dot")
        ),
        row=1, col=1, secondary_y=False
    )

    fig.add_trace(
        go.Scatter(
            x=time_index,
            y=ema_slow,
            name="EMA Slow",
            mode="lines",
            line=dict(color="red", width=1, dash="dash")
        ),
        row=1, col=1, secondary_y=False
    )

    fig.update_yaxes(
        title_text="EXO-Ψ (0–100)",
        range=[0, 100],
        row=1, col=1, secondary_y=False
    )

    # --- Row 2: Composites ----------------------------------------
    fig.add_trace(
        go.Scatter(
            x=time_index,
            y=screen_comp,
            name="Screen Composite",
            mode="lines",
            line=dict(color="magenta", width=1)
        ),
```

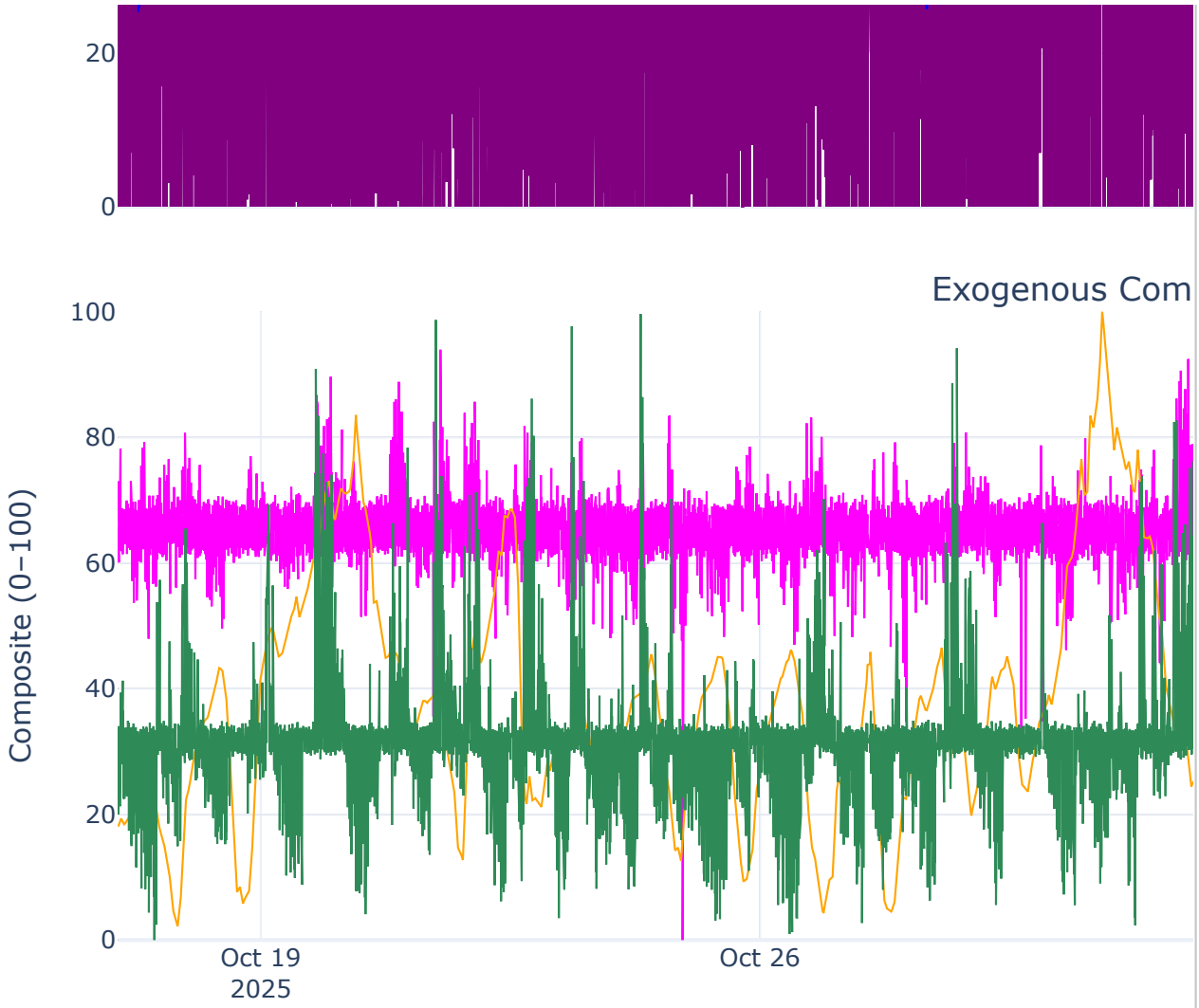
```
        row=2, col=1
    )

    fig.add_trace(
        go.Scatter(
            x=time_index,
            y=weather_comp,
            name="Weather Composite",
            mode="lines",
            line=dict(color="orange", width=1)
        ),
        row=2, col=1
    )

    fig.add_trace(
        go.Scatter(
            x=time_index,
            y=physio_comp,
            name="Physio Composite",
            mode="lines",
            line=dict(color="seagreen", width=1)
        ),
        row=2, col=1
    )

    fig.update_yaxes(
        title_text="Composite (0–100)",
        range=[0, 100],
        row=2, col=1
    )

    # --- Global Layout (Light Mode / Aesthetic) ---------------------
    fig.update_layout(
        title=dict(
            text="EXO-HYPERMIND Quant Dashboard – Interactive Light-Mode \
            x=0.5,
            xanchor="center",
            font=dict(size=20)
        ),
        template="plotly_white",       # LIGHT MODE
        height=900,
        width=1400,
        legend=dict(
            orientation="h",
            yanchor="bottom",
```

```
        y=1.02,
        xanchor="left",
        x=0
    ),
    margin=dict(l=60, r=40, t=90, b=60),
    hovermode="x unified"
)

fig.update_xaxes(title_text="Time", row=2, col=1)

# Show in Colab / Jupyter
fig.show()

# Save as an HTML "website"
html_path = "exo_hypermind_dashboard_light.html"
fig.write_html(html_path)
print(f"\nDashboard saved to: {html_path}\nYou can download it from tl
```

```
TensorFlow: 2.19.0
Keras: 3.10.0

Numeric feature columns used for EXO-HYPERMIND:
['Active Energy (kcal)', 'Apple Exercise Time (min)', 'Apple Stand Hou

Scaled feature matrix shape: (49887, 42) (samples, features)

================ TRAINING EXO-HYPERMIND ================

Epoch 1/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 27s 10ms/step - chaos: 0.1005 - loss: 0.3(
Epoch 2/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0039 - loss: 0.204
Epoch 3/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0029 - loss: 0.203
Epoch 4/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0025 - loss: 0.202
Epoch 5/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0022 - loss: 0.202
Epoch 6/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0021 - loss: 0.202
Epoch 7/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0019 - loss: 0.202
Epoch 8/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0019 - loss: 0.202
Epoch 9/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0017 - loss: 0.201
Epoch 10/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0017 - loss: 0.201
```

```
Epoch 11/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0016 - loss: 0.2019
Epoch 12/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0015 - loss: 0.2018
Epoch 13/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0014 - loss: 0.2017
Epoch 14/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0013 - loss: 0.2016
Epoch 15/15
780/780 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - chaos: 0.0012 - loss: 0.2015
195/195 ━━━━━━━━━━━━━━━━━━━━ 18s 51ms/step

Raw Ψ stats -> min: -2.0035513e-05 max: -1.0798103e-05 mean: -1.7089591

Sample EXO-Ψ values (0–100):
0    65.928368
1    65.639748
2    21.728737
3    46.838413
4    86.461403
Name: EXO_PSI, dtype: float32

Selected columns:
Screen : ['Environmental Audio Exposure (dBASPL)', 'Headphone Audio Exp
Weather: ['temperature', 'humidity', 'rain', 'wind_speed']
Physio : ['Heart Rate [Min] (count/min)', 'Heart Rate [Max] (count/min)

================ BUILDING DASHBOARD (LIGHT MODE) ================
```






```
Dashboard saved to: exo_hypermind_dashboard_light.html
You can download it from the Colab file browser and open in a browser.
```

```
# ============================================================
# EXO-HYPERMIND 2.0  –  CLEAN, LIGHT-MODE, COLAB-READY
# ============================================================

# If plotly isn't available (usually is in Colab), uncomment:
!pip install -q plotly tensorflow scikit-learn pandas numpy

import os
import numpy as np
import pandas as pd
```

```
import tensorflow as tf
from tensorflow.keras import layers, models

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

import plotly.graph_objects as go
from plotly.subplots import make_subplots

# ------------------------------------------------------------
# 1. LOAD DATA
# ------------------------------------------------------------
CSV_PATH = "Multiset-Dataset.csv"

assert os.path.exists(CSV_PATH), (
    f"CSV file '{CSV_PATH}' not found. "
    f"Upload it to the Colab working directory."
)

df = pd.read_csv(CSV_PATH)

# Basic sanity checks
assert "Date/Time" in df.columns, "Expected a 'Date/Time' column in tl

df["Date/Time"] = pd.to_datetime(df["Date/Time"], errors="coerce")
df = df.sort_values("Date/Time").reset_index(drop=True)
df = df.dropna(subset=["Date/Time"])

print("✅ Loaded data:")
print("   Rows:", len(df))
print("   Columns:", len(df.columns))

# ------------------------------------------------------------
# 2. AUTO-DETECT FEATURE GROUPS & BUILD COMPOSITES
# ------------------------------------------------------------

def cols_with(df, keywords):
    """Return columns whose name contains *any* of the keywords."""
    out = []
    for c in df.columns:
        name = c.lower()
        if any(k in name for k in keywords):
            out.append(c)
    return out
```

```
# Try to detect groups by loose keywords
screen_cols = cols_with(df, ["screen", "display", "headphone", "phone"
weather_cols = cols_with(df, ["temp", "humid", "rain", "wind", "weathe
physio_cols = cols_with(df, ["heart rate", "hrv", "spo2", "oxygen", "h

# If something is empty, fall back to numeric columns
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()

if not screen_cols:
    screen_cols = numeric_cols[:3]   # fallback
if not weather_cols:
    weather_cols = numeric_cols[3:7] # fallback
if not physio_cols:
    physio_cols = numeric_cols[7:12] # fallback

print("\n📊 Selected groups:")
print("   Screen :", screen_cols)
print("   Weather:", weather_cols)
print("   Physio :", physio_cols)

def composite_from_cols(df, cols, name):
    """Convert a group of columns to a 0–100 composite."""
    sub = df[cols].copy().astype("float32")
    # Min-max scale per column
    sub_norm = (sub - sub.min()) / (sub.max() - sub.min() + 1e-8)
    comp = sub_norm.mean(axis=1) * 100.0
    df[name] = comp
    return comp

screen_comp  = composite_from_cols(df, screen_cols,  "Screen_Composite
weather_comp = composite_from_cols(df, weather_cols, "Weather_Composit
physio_comp  = composite_from_cols(df, physio_cols,  "Physio_Composite

# Keep only numeric features (including composites)
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()

# ------------------------------------------------------------
# 3. BUILD MODELING DATA: PREDICT NEXT-STEP PHYSIO COMPOSITE
# ------------------------------------------------------------

# Target is next-timestep Physio_Composite
df["Target_Physio_Next"] = df["Physio_Composite"].shift(-1)
model_df = df.dropna(subset=["Target_Physio_Next"]).reset_index(drop=T

feature_cols = numeric_cols.copy()
```

```
# Avoid directly leaking the target-next column as a feature
if "Target_Physio_Next" in feature_cols:
    feature_cols.remove("Target_Physio_Next")

X = model_df[feature_cols].to_numpy().astype("float32")
y = model_df["Target_Physio_Next"].to_numpy().astype("float32")

time_index = model_df["Date/Time"]

print("\n🧱 Modeling set:")
print("   Features shape:", X.shape)
print("   Target shape  :", y.shape)

# Scale X
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train/validation split (time-aware: no shuffle)
split_idx = int(len(X_scaled) * 0.8)
X_train, X_val = X_scaled[:split_idx], X_scaled[split_idx:]
y_train, y_val = y[:split_idx], y[split_idx:]

print("\n📐 Train/Val split:")
print("   Train:", X_train.shape[0], "samples")
print("   Val  :", X_val.shape[0], "samples")

# ------------------------------------------------------------
# 4. DEFINE A STABLE BUT NICE MODEL (EXO-HYPERMIND CORE)
# ------------------------------------------------------------

num_features = X_scaled.shape[1]

inputs = layers.Input(shape=(num_features,), name="features")

x = layers.Dense(128, activation="swish")(inputs)
x = layers.LayerNormalization()(x)
x = layers.Dropout(0.15)(x)

# Residual block
res = layers.Dense(128, activation="swish")(x)
res = layers.Dense(128, activation="linear")(res)
x = layers.Add()([x, res])
x = layers.LayerNormalization()(x)
x = layers.Dropout(0.15)(x)
```

```
x = layers.Dense(64, activation="swish")(x)
x = layers.Dense(32, activation="swish")(x)

outputs = layers.Dense(1, activation="linear", name="physio_next_pred'

exo_model = models.Model(inputs=inputs, outputs=outputs, name="EXO_HYI

exo_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss="mae",
    metrics=["mae"],
)

print("\n🧠 Model summary:")
exo_model.summary(line_length=120)

# ------------------------------------------------------------
# 5. TRAIN MODEL
# ------------------------------------------------------------

EPOCHS = 20
BATCH_SIZE = 64

print("\n🚀 Training EXO-HYPERMIND (predicting next-step Physio Compo
history = exo_model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    verbose=1,
)

# ------------------------------------------------------------
# 6. BUILD EXO-Ψ INDEX (0–100) + EMAs
# ------------------------------------------------------------

y_pred = exo_model.predict(X_scaled, verbose=0).flatten()
exo_raw = y_pred.copy()

exo_min, exo_max = float(exo_raw.min()), float(exo_raw.max())
EXO_PSI = 100.0 * (exo_raw - exo_min) / (exo_max - exo_min + 1e-8)

model_df["EXO_PSI"] = EXO_PSI

# EMAs for smoother regimes
```

```
model_df["EXO_EMA_Fast"] = model_df["EXO_PSI"].ewm(span=96, min_perio
model_df["EXO_EMA_Slow"] = model_df["EXO_PSI"].ewm(span=480, min_peri

print("\n📈 EXO-Ψ stats (after scaling):")
print("   min :", model_df['EXO_PSI'].min())
print("   max :", model_df['EXO_PSI'].max())
print("   mean:", model_df['EXO_PSI'].mean())
print("   std :", model_df['EXO_PSI'].std())

# ------------------------------------------------------------
# 7. PREP DATA FOR DASHBOARD
# ------------------------------------------------------------

t = time_index

exo_series = model_df["EXO_PSI"]
ema_fast   = model_df["EXO_EMA_Fast"]
ema_slow   = model_df["EXO_EMA_Slow"]

screen_series  = model_df["Screen_Composite"]
weather_series = model_df["Weather_Composite"]
physio_series  = model_df["Physio_Composite"]

# ------------------------------------------------------------
# 8. BUILD INTERACTIVE LIGHT-MODE DASHBOARD (PLOTLY)
# ------------------------------------------------------------

fig = make_subplots(
    rows=2, cols=1,
    shared_xaxes=True,
    vertical_spacing=0.06,
    row_heights=[0.6, 0.4],
    subplot_titles=(
        "EXO-Ψ Index with EMAs (0–100)",
        "Screen / Weather / Physio Composites (0–100)",
    )
)

# ----- ROW 1: EXO-Ψ + EMAs -----
fig.add_trace(
    go.Scatter(
        x=t, y=exo_series,
        mode="lines",
        name="EXO-Ψ",
        line=dict(color="purple", width=1.2),
```

```
            hovertemplate="Time=%{x}<br>EXO-Ψ=%{y:.2f}<extra></extra>",
        ),
        row=1, col=1,
    )

    fig.add_trace(
        go.Scatter(
            x=t, y=ema_fast,
            mode="lines",
            name="EMA Fast",
            line=dict(color="dodgerblue", width=1, dash="dot"),
            hovertemplate="Time=%{x}<br>EMA Fast=%{y:.2f}<extra></extra>",
        ),
        row=1, col=1,
    )

    fig.add_trace(
        go.Scatter(
            x=t, y=ema_slow,
            mode="lines",
            name="EMA Slow",
            line=dict(color="orange", width=1, dash="dash"),
            hovertemplate="Time=%{x}<br>EMA Slow=%{y:.2f}<extra></extra>",
        ),
        row=1, col=1,
    )

    # ----- ROW 2: EXOGENOUS COMPOSITES -----
    fig.add_trace(
        go.Scatter(
            x=t, y=screen_series,
            mode="lines",
            name="Screen Composite",
            line=dict(color="magenta", width=1),
            opacity=0.8,
            hovertemplate="Time=%{x}<br>Screen=%{y:.2f}<extra></extra>",
        ),
        row=2, col=1,
    )

    fig.add_trace(
        go.Scatter(
            x=t, y=weather_series,
            mode="lines",
            name="Weather Composite",
```

```
            line=dict(color="teal", width=1),
            opacity=0.8,
            hovertemplate="Time=%{x}<br>Weather=%{y:.2f}<extra></extra>",
        ),
        row=2, col=1,
    )

    fig.add_trace(
        go.Scatter(
            x=t, y=physio_series,
            mode="lines",
            name="Physio Composite",
            line=dict(color="goldenrod", width=1),
            opacity=0.8,
            hovertemplate="Time=%{x}<br>Physio=%{y:.2f}<extra></extra>",
        ),
        row=2, col=1,
    )

    # ----- LAYOUT: LIGHT MODE & PRETTY -----
    fig.update_layout(
        template="plotly_white",
        title=dict(
            text="EXO-HYPERMIND Quant Dashboard — Interactive Light-Mode \
            x=0.5,
            xanchor="center",
            font=dict(size=20, family="Helvetica, Arial"),
        ),
        height=900,
        width=1400,
        legend=dict(
            orientation="h",
            yanchor="bottom",
            y=1.02,
            xanchor="left",
            x=0,
            font=dict(size=11),
        ),
        margin=dict(l=60, r=40, t=80, b=60),
    )

    fig.update_yaxes(
        title_text="EXO-Ψ (0–100)",
        range=[0, 100],
        row=1, col=1,
```

```
)
fig.update_yaxes(
    title_text="Exogenous Metrics (0–100)",
    range=[0, 100],
    row=2, col=1,
)

fig.update_xaxes(
    title_text="Time",
    row=2, col=1,
    rangeslider=dict(visible=True),
    rangeselector=dict(
        buttons=list([
            dict(count=1, label="1d", step="day", stepmode="backward"
            dict(count=3, label="3d", step="day", stepmode="backward"
            dict(count=7, label="1w", step="day", stepmode="backward"
            dict(step="all", label="All"),
        ])
    ),
)

# Show in notebook
fig.show()

# Save as standalone HTML "website"
OUTPUT_HTML = "exo_hypermind_dashboard_light.html"
fig.write_html(OUTPUT_HTML, include_plotlyjs="cdn")
print(f"\n💾 Dashboard saved as: {OUTPUT_HTML}")
print("   You can download it from the Colab file browser and open it
```

```
✅ Loaded data:
   Rows: 49887
   Columns: 43

📊 Selected groups:
   Screen : ['Headphone Audio Exposure (dBASPL)', 'screen_on']
   Weather: ['temperature', 'humidity', 'rain', 'wind_speed']
   Physio : ['Blood Oxygen Saturation (%)', 'Heart Rate [Min] (count/

🧱 Modeling set:
   Features shape: (49886, 45)
   Target shape  : (49886,)

📐 Train/Val split:
   Train: 39908 samples
   Val  : 9978 samples
```

🧠 Model summary:

Model: "EXO_HYPERMIND_CORE"

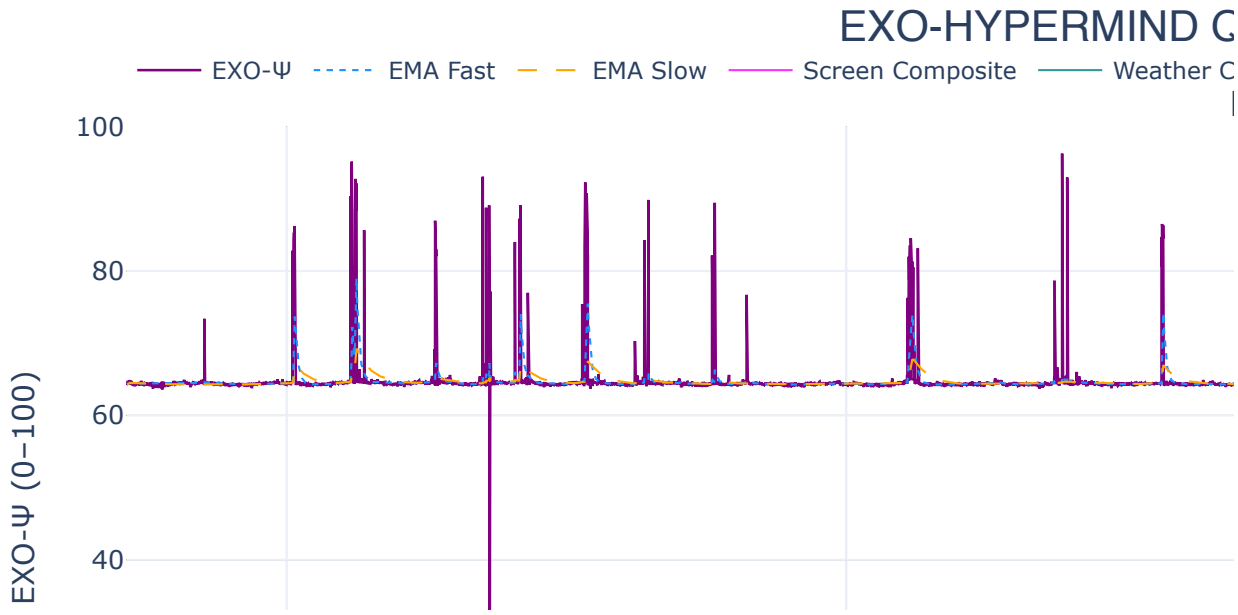
Layer (type)	Output Shape
features (InputLayer)	(None, 45)
dense_90 (Dense)	(None, 128)
layer_normalization_9 (LayerNormalization)	(None, 128)
dropout_10 (Dropout)	(None, 128)
dense_91 (Dense)	(None, 128)
dense_92 (Dense)	(None, 128)
add_13 (Add)	(None, 128)
layer_normalization_10 (LayerNormalization)	(None, 128)
dropout_11 (Dropout)	(None, 128)
dense_93 (Dense)	(None, 64)
dense_94 (Dense)	(None, 32)
physio_next_pred (Dense)	(None, 1)

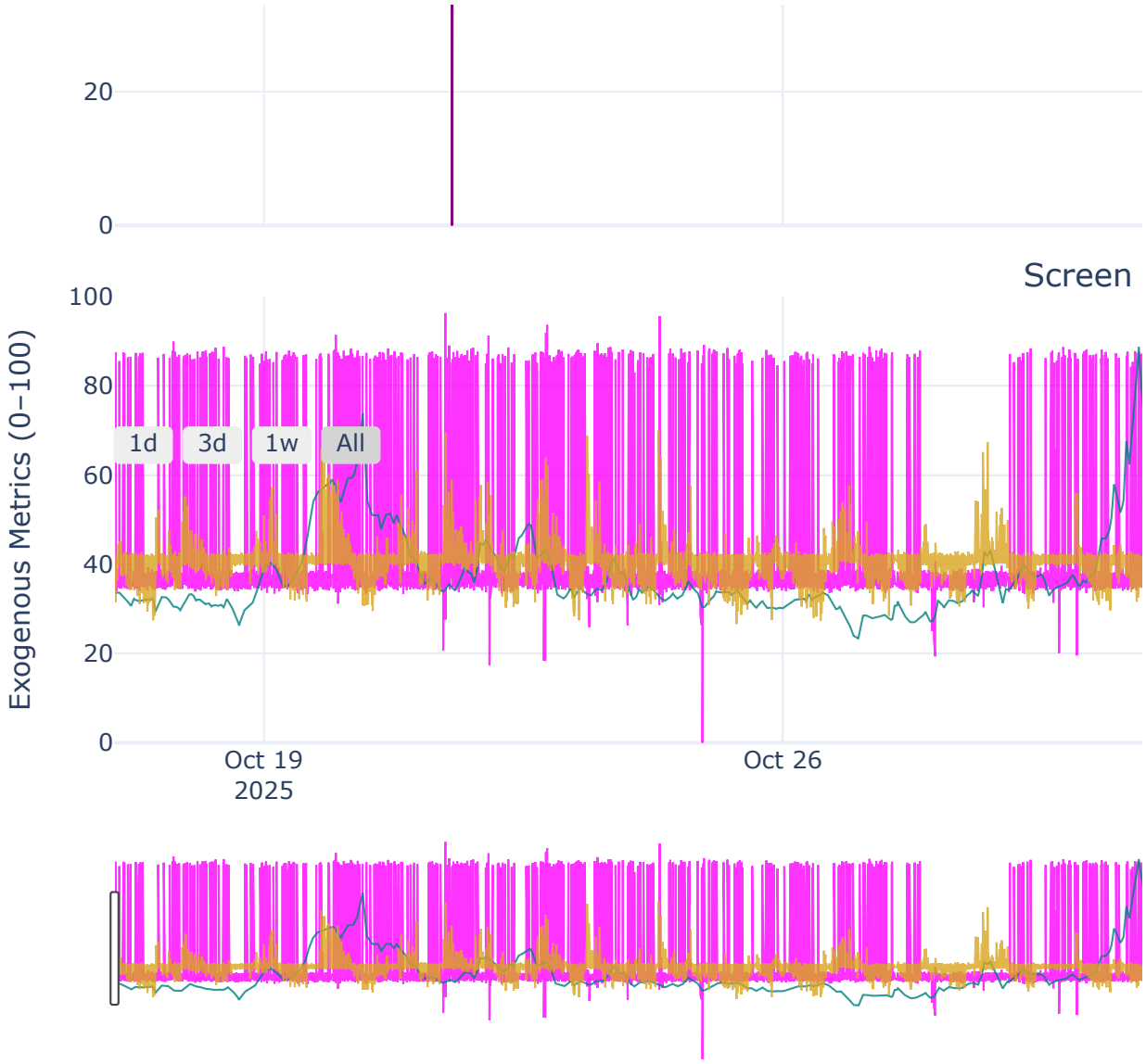
Total params: 49,793 (194.50 KB)
Trainable params: 49,793 (194.50 KB)
Non-trainable params: 0 (0.00 B)

🚀 Training EXO-HYPERMIND (predicting next-step Physio Composite)...

```
Epoch 1/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 9s 8ms/step - loss: 7.5805 - mae: 7.5805
Epoch 2/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.8569 - mae: 1.8569
Epoch 3/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.5939 - mae: 1.5939
Epoch 4/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.4331 - mae: 1.4331
Epoch 5/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.3284 - mae: 1.3284
Epoch 6/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2854 - mae: 1.2854
Epoch 7/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2605 - mae: 1.2605
```

```
Epoch 8/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2382 - mae: 1.2382
Epoch 9/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2368 - mae: 1.2368
Epoch 10/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2310 - mae: 1.2310
Epoch 11/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2291 - mae: 1.2291
Epoch 12/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2236 - mae: 1.2236
Epoch 13/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2213 - mae: 1.2213
Epoch 14/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2124 - mae: 1.2124
Epoch 15/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2076 - mae: 1.2076
Epoch 16/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2113 - mae: 1.2113
Epoch 17/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2091 - mae: 1.2091
Epoch 18/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.1987 - mae: 1.1987
Epoch 19/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2033 - mae: 1.2033
Epoch 20/20
624/624 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.2021 - mae: 1.2021

📈 EXO-Ψ stats (after scaling):
   min : 0.0
   max : 100.00000762939453
   mean: 64.635315
   std : 2.209606409072876
```





💾 Dashboard saved as: exo_hypermind_dashboard_light.html
You can download it from the Colab file browser and open it in any

```
# ============================================================
# EXO-HYPERMIND 2.1  (LSTM + Light-Mode Quant Dashboard)
# Sampling: ~1 minute
# ============================================================

# If you're in Colab, make the output scrollable & wide
from IPython.display import display, HTML
display(HTML("<style>.container { width: 95% !important; }</style>"))
```

```
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import StandardScaler
import plotly.graph_objects as go
from plotly.subplots import make_subplots

np.random.seed(42)
tf.random.set_seed(42)

# ------------------------------------------------------------
# 1. LOAD DATA
# ------------------------------------------------------------
CSV_PATH = "Multiset-Dataset.csv"  # make sure this file is in your Col

assert os.path.exists(CSV_PATH), f"CSV not found at {CSV_PATH}"

df = pd.read_csv(CSV_PATH)
assert "Date/Time" in df.columns, "Expected a 'Date/Time' column in the

df["Date/Time"] = pd.to_datetime(df["Date/Time"])
df = df.sort_values("Date/Time").reset_index(drop=True)

print("\n✅ Loaded dataset")
print(df.head(3))
print("\nColumns:\n", list(df.columns))

# ------------------------------------------------------------
# 2. DEFINE COLUMN GROUPS (SCREEN / WEATHER / PHYSIO)
#    We only use columns that actually exist in your file.
# ------------------------------------------------------------

screen_candidates = [
    "Headphone Audio Exposure (dBASPL)",
    "screen_on",
    "Screen Time (min)",
]

weather_candidates = [
    "temperature",
    "humidity",
    "rain",
```

```
    "wind_speed",
    "Temperature (°C)",
    "Humidity (%)",
    "Rainfall",
    "Wind Speed",
]

physio_candidates = [
    "Heart Rate [Min] (count/min)",
    "Heart Rate [Max] (count/min)",
    "Heart Rate [Avg] (count/min)",
    "Heart Rate Variability (ms)",
    "Resting Heart Rate (count/min)",
]

screen_cols = [c for c in screen_candidates if c in df.columns]
weather_cols = [c for c in weather_candidates if c in df.columns]
physio_cols = [c for c in physio_candidates if c in df.columns]

print("\nSelected columns:")
print(" Screen :", screen_cols)
print(" Weather:", weather_cols)
print(" Physio :", physio_cols)

assert len(physio_cols) > 0, "Need at least one physio column to model

# ------------------------------------------------------------
# 3. COMPOSITE METRICS (0–100) FOR SCREEN / WEATHER / PHYSIO
# ------------------------------------------------------------
def minmax_0_100(series: pd.Series) -> pd.Series:
    s = series.astype(float)
    return 100.0 * (s - s.min()) / (s.max() - s.min() + 1e-8)

if screen_cols:
    screen_comp = minmax_0_100(df[screen_cols].mean(axis=1))
else:
    screen_comp = pd.Series(np.zeros(len(df)), index=df.index)

if weather_cols:
    weather_comp = minmax_0_100(df[weather_cols].mean(axis=1))
else:
    weather_comp = pd.Series(np.zeros(len(df)), index=df.index)

# physio composite will also be the TARGET
physio_raw = df[physio_cols].astype(float)
```

```
physio_z = (physio_raw - physio_raw.mean()) / (physio_raw.std() + 1e-8)
physio_comp = minmax_0_100(physio_z.mean(axis=1))

df["Screen_Composite"] = screen_comp
df["Weather_Composite"] = weather_comp
df["Physio_Composite"] = physio_comp

# ------------------------------------------------------------
# 4. BUILD SEQUENCES FOR LSTM (1-MIN SAMPLING → USE 60-MIN WINDOW)
# ------------------------------------------------------------
WINDOW = 60  # 60 minutes

feature_cols = []
feature_cols += screen_cols
feature_cols += weather_cols
feature_cols += physio_cols  # allow model to see recent physio history

# ensure uniqueness
feature_cols = list(dict.fromkeys(feature_cols))

X_all = df[feature_cols].astype(float).values
y_all = physio_comp.values  # target is composite physio

# scale features for model (not for plots)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_all)

def make_sequences(X, y, window):
    X_seq, y_seq = [], []
    for i in range(len(X) - window):
        X_seq.append(X[i:i+window])
        y_seq.append(y[i+window])
    return np.array(X_seq, dtype=np.float32), np.array(y_seq, dtype=np.

X_seq, y_seq = make_sequences(X_scaled, y_all, WINDOW)
time_seq = df["Date/Time"].iloc[WINDOW:].reset_index(drop=True)

print(f"\nSequence data shape: X_seq={X_seq.shape}, y_seq={y_seq.shape}

# ------------------------------------------------------------
# 5. TRAIN / VALIDATION SPLIT (TIME-ORDERED)
# ------------------------------------------------------------
split_idx = int(0.8 * len(X_seq))
X_train, X_val = X_seq[:split_idx], X_seq[split_idx:]
y_train, y_val = y_seq[:split_idx], y_seq[split_idx:]
time_train, time_val = time_seq[:split_idx], time_seq[split_idx:]
```

```
time_train, time_val = time_seq[:split_idx], time_seq[split_idx:]

print(f"Train sequences: {X_train.shape[0]}, Val sequences: {X_val.shap

# ------------------------------------------------------------
# 6. LSTM MODEL (NEXT-STEP PHYSIO PREDICTION)
# ------------------------------------------------------------
tf.keras.backend.clear_session()

model = keras.Sequential(
    [
        layers.Input(shape=(WINDOW, X_seq.shape[-1])),
        layers.LSTM(64, return_sequences=False),
        layers.Dense(32, activation="relu"),
        layers.Dense(1, activation="linear"),
    ]
)

model.compile(optimizer=keras.optimizers.Adam(1e-3), loss="mse")
model.summary()

early_stop = keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=5, restore_best_weights=True
)

print("\n🔥 Training EXO-HYPERMIND LSTM...")
history = model.fit(
    X_train,
    y_train,
    validation_data=(X_val, y_val),
    epochs=20,
    batch_size=64,
    verbose=1,
    callbacks=[early_stop],
)

# ------------------------------------------------------------
# 7. GENERATE PREDICTIONS + EXO-Ψ STABILITY INDEX
# ------------------------------------------------------------
y_pred_all = model.predict(X_seq, batch_size=256).reshape(-1)

abs_err = np.abs(y_seq - y_pred_all)
scale = np.percentile(abs_err, 95) + 1e-8
psi_0_1 = 1.0 - np.clip(abs_err / scale, 0.0, 1.0)
EXO_PSI = 100.0 * psi_0_1
```

```
print(
    "\nEXO-Ψ raw stats:",
    f"min={EXO_PSI.min():.2f}, max={EXO_PSI.max():.2f}, mean={EXO_PSI.m
)

# ------------------------------------------------------------
# 8. SMOOTH EXO-Ψ WITH EMAs
# ------------------------------------------------------------
exo_series = pd.Series(EXO_PSI, index=time_seq)
EMA_FAST_SPAN = 30   # 30-minute EMA
EMA_SLOW_SPAN = 240  # 4-hour EMA

exo_ema_fast = exo_series.ewm(span=EMA_FAST_SPAN).mean()
exo_ema_slow = exo_series.ewm(span=EMA_SLOW_SPAN).mean()

# align exogenous composites to same time index (skip first WINDOW poin
screen_vis = screen_comp.iloc[WINDOW:].reset_index(drop=True)
weather_vis = weather_comp.iloc[WINDOW:].reset_index(drop=True)
physio_vis = physio_comp.iloc[WINDOW:].reset_index(drop=True)

# ------------------------------------------------------------
# 9. BUILD DASHBOARD DATAFRAME
# ------------------------------------------------------------
plot_df = pd.DataFrame(
    {
        "Time": time_seq,
        "EXO_PSI": exo_series.values,
        "EXO_EMA_Fast": exo_ema_fast.values,
        "EXO_EMA_Slow": exo_ema_slow.values,
        "Screen_Composite": screen_vis.values,
        "Weather_Composite": weather_vis.values,
        "Physio_Composite": physio_vis.values,
    }
)

# ------------------------------------------------------------
# 10. PLOTLY LIGHT-MODE QUANT DASHBOARD
# ------------------------------------------------------------
fig = make_subplots(
    rows=3,
    cols=1,
    shared_xaxes=True,
    vertical_spacing=0.04,
    row_heights=[0.4, 0.35, 0.25],
    specs=[
```

```
        specs=[
            [{}],
            [{"secondary_y": True}],
            [{}],
        ],
    )

    # ---- Row 1: EXO-Ψ + EMAs ----------------------------------
    fig.add_trace(
        go.Scatter(
            x=plot_df["Time"],
            y=plot_df["EXO_PSI"],
            mode="lines",
            name="EXO-Ψ",
            line=dict(color="purple", width=1.5),
        ),
        row=1,
        col=1,
    )

    fig.add_trace(
        go.Scatter(
            x=plot_df["Time"],
            y=plot_df["EXO_EMA_Fast"],
            mode="lines",
            name="EMA Fast",
            line=dict(color="dodgerblue", width=1, dash="dot"),
        ),
        row=1,
        col=1,
    )

    fig.add_trace(
        go.Scatter(
            x=plot_df["Time"],
            y=plot_df["EXO_EMA_Slow"],
            mode="lines",
            name="EMA Slow",
            line=dict(color="orange", width=1, dash="dash"),
        ),
        row=1,
        col=1,
    )

    fig.update_yaxes(title_text="EXO-Ψ (0–100)", row=1, col=1)
```

```
    # ---- Row 2: Exogenous composites + EXO-Ψ on secondary axis --
    fig.add_trace(
        go.Scatter(
            x=plot_df["Time"],
            y=plot_df["Screen_Composite"],
            mode="lines",
            name="Screen Composite",
            line=dict(color="mediumblue", width=1),
        ),
        row=2,
        col=1,
        secondary_y=False,
    )

    fig.add_trace(
        go.Scatter(
            x=plot_df["Time"],
            y=plot_df["Weather_Composite"],
            mode="lines",
            name="Weather Composite",
            line=dict(color="darkorange", width=1),
        ),
        row=2,
        col=1,
        secondary_y=False,
    )

    fig.add_trace(
        go.Scatter(
            x=plot_df["Time"],
            y=plot_df["Physio_Composite"],
            mode="lines",
            name="Physio Composite",
            line=dict(color="seagreen", width=1),
        ),
        row=2,
        col=1,
        secondary_y=False,
    )

    fig.add_trace(
        go.Scatter(
            x=plot_df["Time"],
            y=plot_df["EXO_PSI"],
            mode="lines",
```

```
            name="EXO-Ψ (overlay)",
            line=dict(color="rebeccapurple", width=1, dash="dot"),
            opacity=0.6,
        ),
        row=2,
        col=1,
        secondary_y=True,
    )

    fig.update_yaxes(title_text="Exogenous Metrics (0–100)", row=2, col=1,
    fig.update_yaxes(title_text="EXO-Ψ (0–100)", row=2, col=1, secondary_y=

    # ---- Row 3: Scatter relations EXO-Ψ vs exogenous ------------
    fig.add_trace(
        go.Scatter(
            x=plot_df["Screen_Composite"],
            y=plot_df["EXO_PSI"],
            mode="markers",
            name="EXO-Ψ vs Screen",
            marker=dict(color="royalblue", size=4, opacity=0.4),
        ),
        row=3,
        col=1,
    )

    fig.add_trace(
        go.Scatter(
            x=plot_df["Weather_Composite"],
            y=plot_df["EXO_PSI"],
            mode="markers",
            name="EXO-Ψ vs Weather",
            marker=dict(color="darkorange", size=4, opacity=0.4),
        ),
        row=3,
        col=1,
    )

    fig.add_trace(
        go.Scatter(
            x=plot_df["Physio_Composite"],
            y=plot_df["EXO_PSI"],
            mode="markers",
            name="EXO-Ψ vs Physio",
            marker=dict(color="seagreen", size=4, opacity=0.4),
        ),
```

```
        row=3,
        col=1,
    )

    fig.update_xaxes(title_text="Composite Metric (0–100)", row=3, col=1)
    fig.update_yaxes(title_text="EXO-Ψ (0–100)", row=3, col=1)

    # ---- Layout / aesthetics ------------------------------------
    fig.update_layout(
        template="plotly_white",
        title=(
            "EXO-HYPERMIND Quant Dashboard — Interactive Light-Mode View<br
            "<sup>Top: EXO-Ψ & EMAs | Middle: Exogenous metrics + overlay E
            "Bottom: Scatter relations</sup>"
        ),
        width=1400,
        height=900,
        legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="lef
        margin=dict(l=60, r=30, t=80, b=40),
    )

    fig.update_xaxes(showgrid=True)
    fig.update_yaxes(showgrid=True)

    fig.show()

    # save as standalone HTML
    OUTPUT_HTML = "exo_hypermind_dashboard_light.html"
    fig.write_html(OUTPUT_HTML)
    print(f"\n💾 Dashboard saved to: {OUTPUT_HTML}")
```

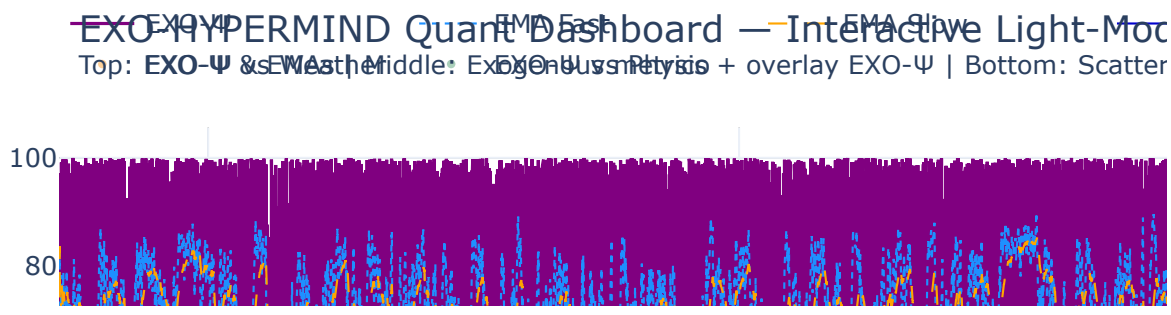
```
✅ Loaded dataset
             Date/Time  Active Energy (kcal)  Apple Exercise Time (min
0 2025-10-17 00:00:00              0.131598                     1.01254
1 2025-10-17 00:01:00              0.264936                     1.00622
2 2025-10-17 00:02:00              0.264833                     1.02294

   Apple Stand Hour (count)  Apple Stand Time (min)  \
0                   1.00000                0.200000
1                   1.01932                0.200000
2                   1.00000                0.200842

   Blood Oxygen Saturation (%)  Environmental Audio Exposure (dBASPL)
0                    98.086993                              62.595498
1                    98.300413                              63.604034
2                    98.002632                              63.716417
```

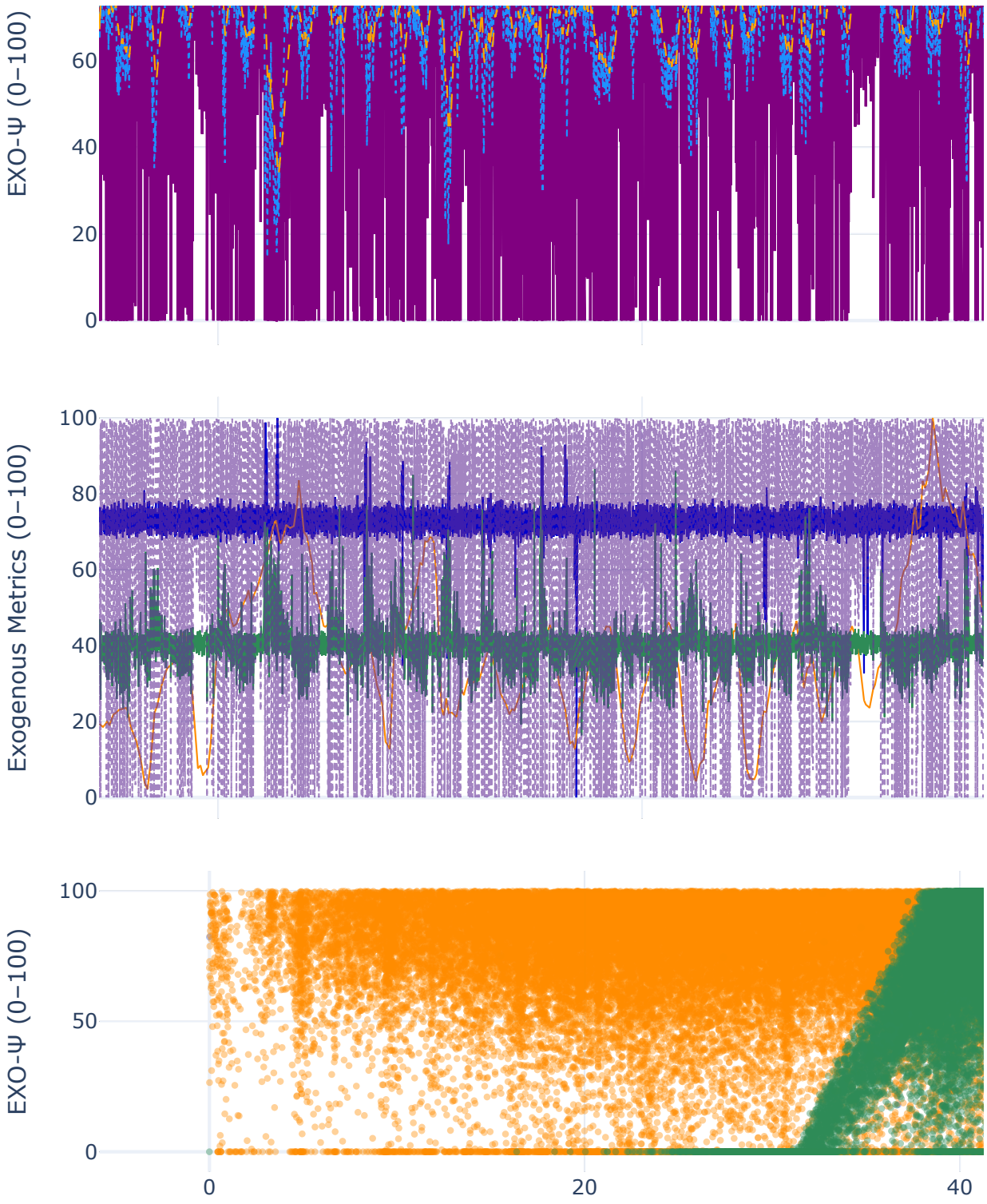
```
   Flights Climbed (count)  Headphone Audio Exposure (dBASPL)  \
0                 0.575021                          67.264885
1                 0.605586                          67.750293
2                 0.623950                          67.245333

   Heart Rate [Min] (count/min)  ...  Walking Heart Rate Average (cou
0                     90.932521  ...                             115
1                     88.119720  ...                             115
2                     86.321881  ...                             117

   Walking Speed (mi/hr)  Walking Step Length (in)  upload_byte_count
0               2.698167                 26.447500            1101729
1               2.750439                 26.292207             889484
2               2.738717                 26.454021             969855

   screen_on  temperature  feels_like   humidity  rain  wind_speed
0          1        11.00    6.800000  48.000000   0.0   14.600000
1          0        10.99    6.788333  48.016667   0.0   14.611667
2          0        10.98    6.776667  48.033333   0.0   14.623333

[3 rows x 43 columns]

Columns:
 ['Date/Time', 'Active Energy (kcal)', 'Apple Exercise Time (min)', '

Selected columns:
 Screen : ['Headphone Audio Exposure (dBASPL)', 'screen_on']
 Weather: ['temperature', 'humidity', 'rain', 'wind_speed']
 Physio : ['Heart Rate [Min] (count/min)', 'Heart Rate [Max] (count/m

Sequence data shape: X_seq=(49827, 60, 11), y_seq=(49827,)
Train sequences: 39861, Val sequences: 9966
Model: "sequential"
```

Layer (type)	Output Shape	Pa
lstm (LSTM)	(None, 64)	1
dense (Dense)	(None, 32)	
dense_1 (Dense)	(None, 1)	

```
 Total params: 21,569 (84.25 KB)
 Trainable params: 21,569 (84.25 KB)
 Non-trainable params: 0 (0.00 B)

🔥 Training EXO-HYPERMIND LSTM...
Epoch 1/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 456.1394 - val_loss:
```

```
Epoch 2/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 15.4708 - val_loss:
Epoch 3/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 13.9293 - val_loss:
Epoch 4/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 13.7853 - val_loss:
Epoch 5/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.5679 - val_loss:
Epoch 6/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.3166 - val_loss:
Epoch 7/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.1974 - val_loss:
Epoch 8/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.1134 - val_loss:
Epoch 9/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.0530 - val_loss:
Epoch 10/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.0066 - val_loss:
Epoch 11/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.9692 - val_loss:
Epoch 12/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.9379 - val_loss:
Epoch 13/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.9111 - val_loss:
Epoch 14/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.8865 - val_loss:
Epoch 15/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.8636 - val_loss:
Epoch 16/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.8404 - val_loss:
Epoch 17/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.8244 - val_loss:
Epoch 18/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.7992 - val_loss:
Epoch 19/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.8356 - val_loss:
Epoch 20/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.7741 - val_loss:
195/195 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step

EXO-Ψ raw stats: min=0.00, max=100.00, mean=72.31
```





💾 Dashboard saved to: exo_hypermind_dashboard_light.html

```
# ============================================================
# EXO-HYPERMIND (Readable Light-Mode Dashboard Version, FIXED TITLE/LI
```

```
# ============================================================

from IPython.display import display, HTML
display(HTML("<style>.container { width: 95% !important; }</style>"))

import os
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import StandardScaler
import plotly.graph_objects as go
from plotly.subplots import make_subplots

np.random.seed(42)
tf.random.set_seed(42)

# ------------------------------------------------------------
# 1. LOAD DATA
# ------------------------------------------------------------
CSV_PATH = "Multiset-Dataset.csv"   # <-- keep this name

assert os.path.exists(CSV_PATH), f"CSV not found at {CSV_PATH}"

df = pd.read_csv(CSV_PATH)
assert "Date/Time" in df.columns, "Expected a 'Date/Time' column in tl

df["Date/Time"] = pd.to_datetime(df["Date/Time"])
df = df.sort_values("Date/Time").reset_index(drop=True)

print("\n✅ Loaded dataset")
print(df.head(3))
print("\nColumns:\n", list(df.columns))

# ------------------------------------------------------------
# 2. CHOOSE COLUMN GROUPS PRESENT IN YOUR FILE
# ------------------------------------------------------------
screen_candidates = [
    "Headphone Audio Exposure (dBASPL)",
    "screen_on",
    "Screen Time (min)",
]
weather_candidates = [
    "temperature", "humidity", "rain", "wind_speed",
```

```
    "Temperature (°C)", "Humidity (%)", "Rainfall", "Wind Speed",
]
physio_candidates = [
    "Heart Rate [Min] (count/min)",
    "Heart Rate [Max] (count/min)",
    "Heart Rate [Avg] (count/min)",
    "Heart Rate Variability (ms)",
    "Resting Heart Rate (count/min)",
]

screen_cols  = [c for c in screen_candidates  if c in df.columns]
weather_cols = [c for c in weather_candidates if c in df.columns]
physio_cols  = [c for c in physio_candidates  if c in df.columns]

print("\nSelected columns actually in file:")
print(" Screen :", screen_cols)
print(" Weather:", weather_cols)
print(" Physio :", physio_cols)
assert len(physio_cols) > 0, "Need at least one physio column."

# ------------------------------------------------------------
# 3. COMPOSITE METRICS (0–100)
# ------------------------------------------------------------
def minmax_0_100(series: pd.Series) -> pd.Series:
    s = series.astype(float)
    return 100.0 * (s - s.min()) / (s.max() - s.min() + 1e-8)

if screen_cols:
    screen_comp = minmax_0_100(df[screen_cols].mean(axis=1))
else:
    screen_comp = pd.Series(np.zeros(len(df)), index=df.index)

if weather_cols:
    weather_comp = minmax_0_100(df[weather_cols].mean(axis=1))
else:
    weather_comp = pd.Series(np.zeros(len(df)), index=df.index)

# Physio composite (this is what we try to predict next-step)
physio_raw = df[physio_cols].astype(float)
physio_z = (physio_raw - physio_raw.mean()) / (physio_raw.std() + 1e-8
physio_comp = minmax_0_100(physio_z.mean(axis=1))

df["Screen_Composite"]  = screen_comp
df["Weather_Composite"] = weather_comp
df["Physio_Composite"]  = physio_comp
```

```
# ------------------------------------------------------------
# 4. BUILD SEQUENCES FOR LSTM (60-minute history window)
# ------------------------------------------------------------
WINDOW = 60  # 60 samples ~ 60 minutes

feature_cols = list(dict.fromkeys(screen_cols + weather_cols + physio_
X_all = df[feature_cols].astype(float).values
y_all = physio_comp.values

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_all)

def make_sequences(X, y, window):
    X_seq, y_seq = [], []
    for i in range(len(X) - window):
        X_seq.append(X[i : i + window])
        y_seq.append(y[i + window])
    return np.array(X_seq, dtype=np.float32), np.array(y_seq, dtype=np

X_seq, y_seq = make_sequences(X_scaled, y_all, WINDOW)
time_seq = df["Date/Time"].iloc[WINDOW:].reset_index(drop=True)

print(f"\nSequence shapes: X_seq={X_seq.shape}, y_seq={y_seq.shape}")

# ------------------------------------------------------------
# 5. TRAIN / VAL SPLIT
# ------------------------------------------------------------
split_idx = int(0.8 * len(X_seq))
X_train, X_val = X_seq[:split_idx], X_seq[split_idx:]
y_train, y_val = y_seq[:split_idx], y_seq[split_idx:]
time_train, time_val = time_seq[:split_idx], time_seq[split_idx:]

print(f"Train sequences: {len(X_train)}, Val sequences: {len(X_val)}")

# ------------------------------------------------------------
# 6. LSTM MODEL
# ------------------------------------------------------------
tf.keras.backend.clear_session()

model = keras.Sequential(
    [
        layers.Input(shape=(WINDOW, X_seq.shape[-1])),
        layers.LSTM(64, return_sequences=False),
        layers.Dense(32, activation="relu"),
```

```
        layers.Dense(1, activation="linear"),
    ]
)

model.compile(optimizer=keras.optimizers.Adam(1e-3), loss="mse")
model.summary()

early_stop = keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=5, restore_best_weights=True
)

print("\n🔥 Training LSTM for EXO-Ψ...")
history = model.fit(
    X_train,
    y_train,
    validation_data=(X_val, y_val),
    epochs=20,
    batch_size=64,
    verbose=1,
    callbacks=[early_stop],
)

# ------------------------------------------------------------
# 7. PREDICTIONS + EXO-Ψ STABILITY INDEX
# ------------------------------------------------------------
y_pred_all = model.predict(X_seq, batch_size=256).reshape(-1)
abs_err = np.abs(y_seq - y_pred_all)
scale = np.percentile(abs_err, 95) + 1e-8
psi_0_1 = 1.0 - np.clip(abs_err / scale, 0.0, 1.0)
EXO_PSI = 100.0 * psi_0_1

exo_series = pd.Series(EXO_PSI, index=time_seq)

print(
    "\nEXO-Ψ stats (0-100): "
    f"min={EXO_PSI.min():.2f}, max={EXO_PSI.max():.2f}, mean={EXO_PSI
)

EMA_FAST_SPAN = 30   # 30-min
EMA_SLOW_SPAN = 240  # 4-hour

exo_ema_fast = exo_series.ewm(span=EMA_FAST_SPAN).mean()
exo_ema_slow = exo_series.ewm(span=EMA_SLOW_SPAN).mean()

screen_vis  = screen_comp.iloc[WINDOW:].reset_index(drop=True)
```

```
weather_vis = weather_comp.iloc[WINDOW:].reset_index(drop=True)
physio_vis  = physio_comp.iloc[WINDOW:].reset_index(drop=True)

plot_df = pd.DataFrame(
    {
        "Time": time_seq,
        "EXO_PSI": exo_series.values,
        "EXO_EMA_Fast": exo_ema_fast.values,
        "EXO_EMA_Slow": exo_ema_slow.values,
        "Screen_Composite": screen_vis.values,
        "Weather_Composite": weather_vis.values,
        "Physio_Composite": physio_vis.values,
    }
)

# ------------------------------------------------------------
# 8. DOWNSAMPLE FOR CLEAN VISUAL (10-MIN BARS)
# ------------------------------------------------------------
plot_df_indexed = plot_df.set_index("Time")
plot_df_res = plot_df_indexed.resample("10T").mean().dropna().reset_i
print(f"\nDownsampled for plotting: {len(plot_df_res)} points (10-min

scatter_df = plot_df.sample(
    n=min(3000, len(plot_df)), random_state=42
).reset_index(drop=True)

# ------------------------------------------------------------
# 9. BUILD CLEAN LIGHT-MODE DASHBOARD (FIXED TITLE / LEGEND)
# ------------------------------------------------------------
fig = make_subplots(
    rows=3,
    cols=1,
    shared_xaxes=False,
    vertical_spacing=0.07,
    row_heights=[0.4, 0.35, 0.25],
)

# Row 1: EXO-Ψ EMAs
fig.add_trace(
    go.Scatter(
        x=plot_df_res["Time"],
        y=plot_df_res["EXO_EMA_Fast"],
        mode="lines",
        name="EXO-Ψ EMA Fast (30m)",
        line=dict(color="royalblue", width=2),
```

```
        ),
        row=1,
        col=1,
    )

    fig.add_trace(
        go.Scatter(
            x=plot_df_res["Time"],
            y=plot_df_res["EXO_EMA_Slow"],
            mode="lines",
            name="EXO-Ψ EMA Slow (4h)",
            line=dict(color="firebrick", width=2),
        ),
        row=1,
        col=1,
    )

    fig.update_yaxes(title_text="EXO-Ψ (0–100)", range=[0, 100], row=1, c

    # Row 2: composites
    fig.add_trace(
        go.Scatter(
            x=plot_df_res["Time"],
            y=plot_df_res["Screen_Composite"],
            mode="lines",
            name="Screen Composite",
            line=dict(color="mediumpurple", width=1.5),
        ),
        row=2,
        col=1,
    )
    fig.add_trace(
        go.Scatter(
            x=plot_df_res["Time"],
            y=plot_df_res["Weather_Composite"],
            mode="lines",
            name="Weather Composite",
            line=dict(color="darkorange", width=1.5),
        ),
        row=2,
        col=1,
    )
    fig.add_trace(
        go.Scatter(
            x=plot_df_res["Time"],
```

```
            y=plot_df_res["Physio_Composite"],
            mode="lines",
            name="Physio Composite",
            line=dict(color="seagreen", width=1.5),
        ),
        row=2,
        col=1,
    )
    fig.update_yaxes(
        title_text="Composites (0–100)",
        range=[0, 100],
        row=2,
        col=1,
    )

    # Row 3: scatter
    fig.add_trace(
        go.Scatter(
            x=scatter_df["Screen_Composite"],
            y=scatter_df["EXO_PSI"],
            mode="markers",
            name="EXO–Ψ vs Screen",
            marker=dict(color="mediumpurple", size=4, opacity=0.45),
        ),
        row=3,
        col=1,
    )
    fig.add_trace(
        go.Scatter(
            x=scatter_df["Weather_Composite"],
            y=scatter_df["EXO_PSI"],
            mode="markers",
            name="EXO–Ψ vs Weather",
            marker=dict(color="darkorange", size=4, opacity=0.45),
        ),
        row=3,
        col=1,
    )
    fig.add_trace(
        go.Scatter(
            x=scatter_df["Physio_Composite"],
            y=scatter_df["EXO_PSI"],
            mode="markers",
            name="EXO–Ψ vs Physio",
            marker=dict(color="seagreen", size=4, opacity=0.45),
```

```
        ),
        row=3,
        col=1,
    )

    fig.update_xaxes(title_text="Composite value (0–100)", row=3, col=1)
    fig.update_yaxes(title_text="EXO-Ψ (0–100)", range=[0, 100], row=3, c

    # ---- Layout fix: title and legend ------------------------------
    fig.update_layout(
        template="simple_white",
        title=dict(
            text="EXO-HYPERMIND Quant Dashboard – Clean Light-Mode View",
            x=0.5,
            xanchor="center",
            y=0.96,
            yanchor="top",
            font=dict(size=22),
        ),
        width=1400,
        height=900,
        legend=dict(
            orientation="h",
            yanchor="bottom",
            y=-0.12,           # move legend BELOW the plots
            xanchor="center",
            x=0.5,
            font=dict(size=12),
        ),
        margin=dict(l=70, r=40, t=70, b=140),  # extra bottom room for le
    )

    # small subtitle as annotation (not in legend)
    fig.add_annotation(
        text="Row 1: EXO-Ψ EMAs • Row 2: Screen / Weather / Physio compos
        xref="paper",
        yref="paper",
        x=0.5,
        y=1.03,
        showarrow=False,
        font=dict(size=12),
    )

    fig.update_xaxes(showgrid=True)
    fig.update_yaxes(showgrid=True)
```

```
fig.show()

OUTPUT_HTML = "exo_hypermind_dashboard_light_clean.html"
fig.write_html(OUTPUT_HTML)
print(f"\n💾 Dashboard saved to: {OUTPUT_HTML}")
```

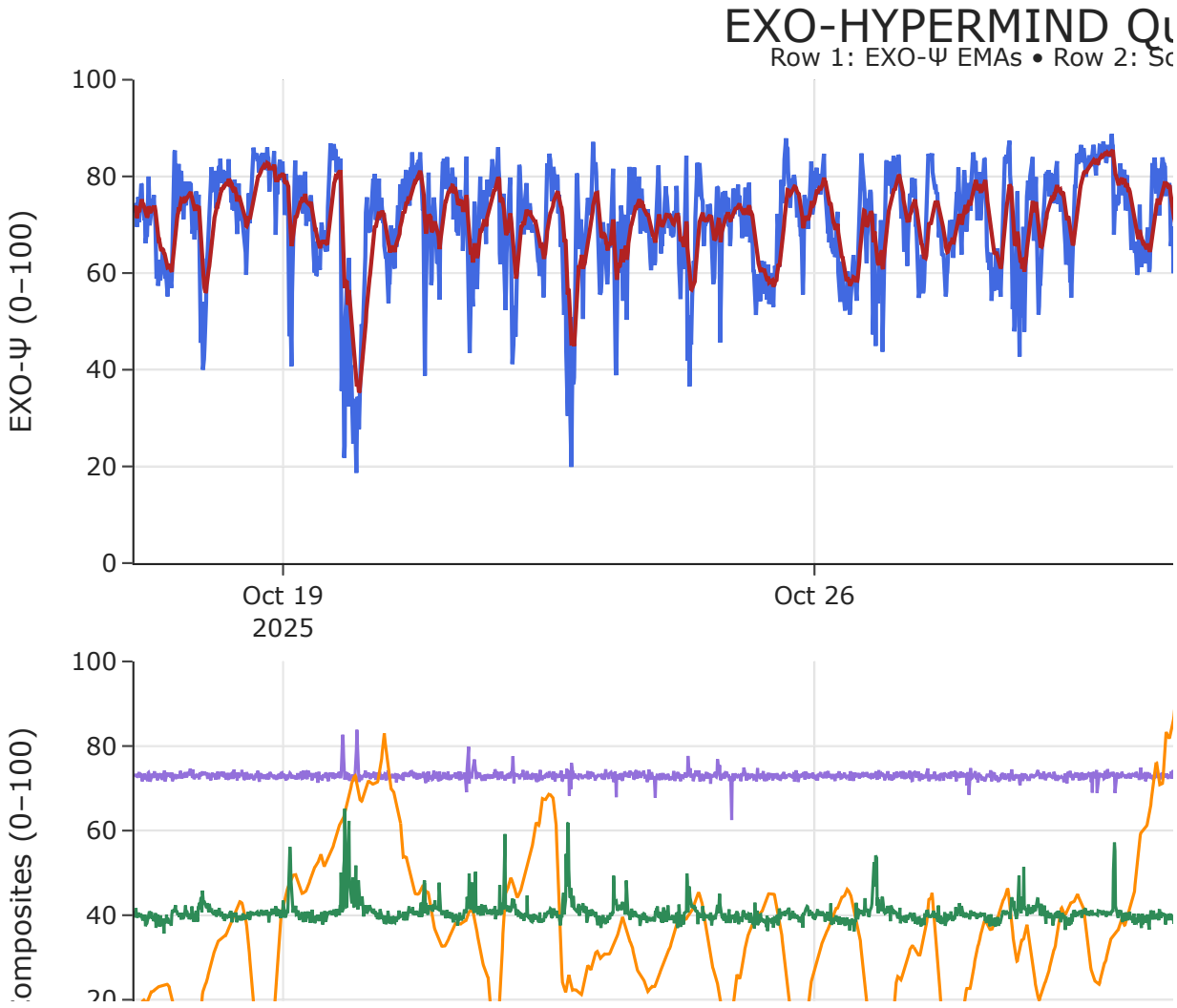
```
✅ Loaded dataset
             Date/Time  Active Energy (kcal)  Apple Exercise Time (min
0 2025-10-17 00:00:00              0.131598                    1.01254
1 2025-10-17 00:01:00              0.264936                    1.00622
2 2025-10-17 00:02:00              0.264833                    1.02294

   Apple Stand Hour (count)  Apple Stand Time (min)  \
0                   1.00000                0.200000
1                   1.01932                0.200000
2                   1.00000                0.200842

   Blood Oxygen Saturation (%)  Environmental Audio Exposure (dBASPL)
0                    98.086993                              62.595498
1                    98.300413                              63.604034
2                    98.002632                              63.716417

   Flights Climbed (count)  Headphone Audio Exposure (dBASPL)  \
0                 0.575021                          67.264885
1                 0.605586                          67.750293
2                 0.623950                          67.245333

   Heart Rate [Min] (count/min)  ...  Walking Heart Rate Average (cou
0                     90.932521  ...                              115
1                     88.119720  ...                              115
2                     86.321881  ...                              117

   Walking Speed (mi/hr)  Walking Step Length (in)  upload_byte_count
0               2.698167                 26.447500            1101729
1               2.750439                 26.292207             889484
2               2.738717                 26.454021             969855

   screen_on  temperature  feels_like   humidity  rain  wind_speed
0          1        11.00    6.800000  48.000000   0.0   14.600000
1          0        10.99    6.788333  48.016667   0.0   14.611667
2          0        10.98    6.776667  48.033333   0.0   14.623333

[3 rows x 43 columns]

Columns:
 ['Date/Time', 'Active Energy (kcal)', 'Apple Exercise Time (min)', '

Selected columns actually in file:
```

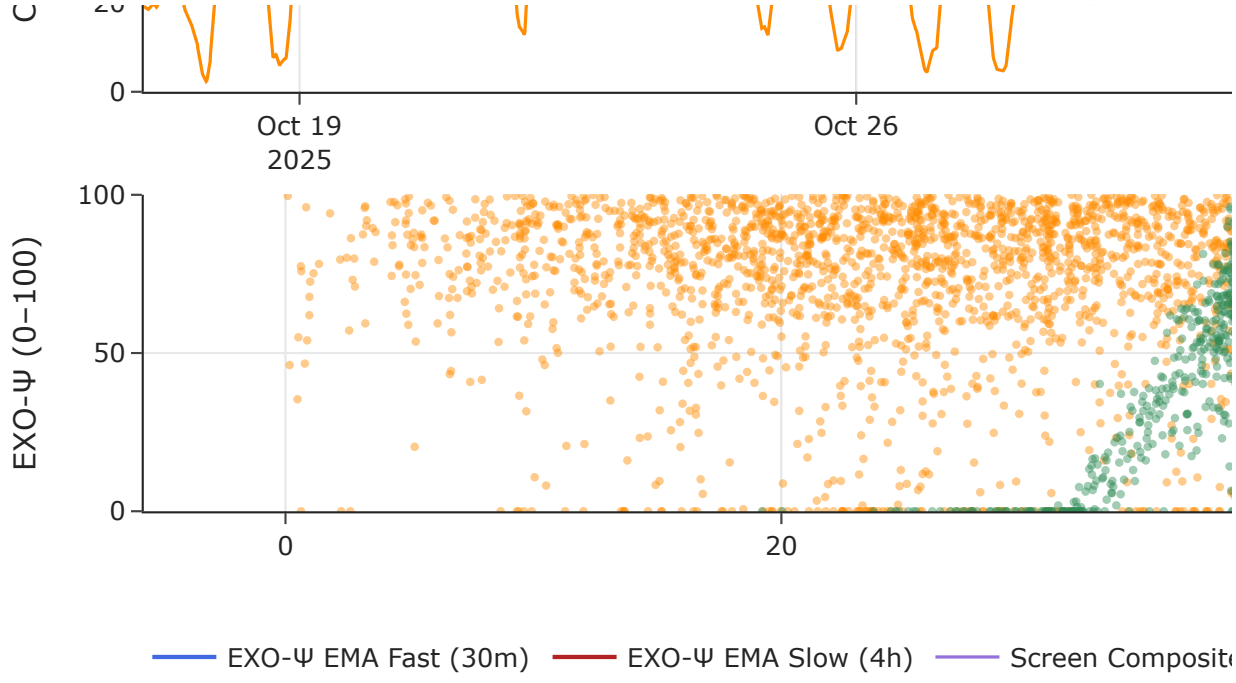
```
 Screen : ['Headphone Audio Exposure (dBASPL)', 'screen_on']
 Weather: ['temperature', 'humidity', 'rain', 'wind_speed']
 Physio : ['Heart Rate [Min] (count/min)', 'Heart Rate [Max] (count/m

Sequence shapes: X_seq=(49827, 60, 11), y_seq=(49827,)
Train sequences: 39861, Val sequences: 9966
Model: "sequential"
```

Layer (type)	Output Shape	Pa
lstm (LSTM)	(None, 64)	1
dense (Dense)	(None, 32)	
dense_1 (Dense)	(None, 1)	

```
 Total params: 21,569 (84.25 KB)
 Trainable params: 21,569 (84.25 KB)
 Non-trainable params: 0 (0.00 B)

🔥 Training LSTM for EXO-Ψ...
Epoch 1/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 523.5505 - val_loss:
Epoch 2/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 16.1632 - val_loss:
Epoch 3/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 13.9307 - val_loss:
Epoch 4/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 13.1794 - val_loss:
Epoch 5/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.8429 - val_loss:
Epoch 6/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.4945 - val_loss:
Epoch 7/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.2441 - val_loss:
Epoch 8/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.1515 - val_loss:
Epoch 9/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.0924 - val_loss:
Epoch 10/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 12.0429 - val_loss:
Epoch 11/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.9959 - val_loss:
Epoch 12/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.9482 - val_loss:
Epoch 13/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.8846 - val_loss:
Epoch 14/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.8279 - val_loss:
Epoch 15/20
```

```
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.7850 - val_loss:
Epoch 16/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.7504 - val_loss:
Epoch 17/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.7186 - val_loss:
Epoch 18/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.6910 - val_loss:
Epoch 19/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.6643 - val_loss:
Epoch 20/20
623/623 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 11.6452 - val_loss:
195/195 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step

EXO-Ψ stats (0–100): min=0.00, max=100.00, mean=72.13

Downsampled for plotting: 4980 points (10-min bars).
/tmp/ipython-input-3696228285.py:202: FutureWarning:

'T' is deprecated and will be removed in a future version, please use
```





💾 Dashboard saved to: exo_hypermind_dashboard_light_clean.html