Jin: Unified Multimodal AI Architecture for Vision-Language, Audio, and Cross-Modal Understanding

Zach Kelling[*]
Hanzo Industries *Lux Industries* *Zoo Labs Foundation*
`research@hanzo.ai`

2025

Abstract

We present Jin, a hypermodal AI architecture that unifies text, vision, audio, and extensible custom modalities through a joint embedding predictive framework. Jin combines three key innovations: (1) modality-specific encoders projecting heterogeneous inputs into a shared latent space with learnable modality embeddings; (2) a diffusion transformer backbone with Mixture of Experts (MoE) routing for efficient cross-modal reasoning; (3) hierarchical zooming JEPA (z-JEPA) for multi-scale visual understanding. Our experiments demonstrate that Jin achieves 82.1% accuracy on ImageNet with zero-shot transfer, outperforms single-modality baselines by 15% on AudioCaps retrieval, and enables novel cross-modal generation tasks. The architecture supports runtime modality extension without retraining the core model, enabling deployment flexibility for diverse applications.

1 Introduction

Human cognition seamlessly integrates information across sensory modalities—we understand speech by combining audio with visual lip movements, interpret scenes by relating objects to their sounds, and communicate using language grounded in physical experience. Current AI systems largely process modalities in isolation, with multimodal models treating cross-modal understanding as a downstream fusion task.

[*]zach@hanzo.ai

Jin (Japanese: UTF8min, "spirit" or "god") introduces a hypermodal architecture where all modalities share a unified latent space from the ground up. The key insight is that joint embedding predictive architectures (JEPA) [1] can be extended beyond single modalities: by training encoders to predict masked representations across modalities, the model learns modality-agnostic features suitable for any downstream task.

Contributions. This paper makes the following contributions:

- A joint embedding space architecture with modality-specific encoders and learnable modality tokens enabling seamless cross-modal fusion.
- A diffusion transformer backbone with Mixture of Experts providing efficient routing for modality-specific and cross-modal computation.
- z-JEPA, a hierarchical zooming extension of JEPA for multi-scale visual understanding with adaptive computation.
- Runtime modality extension enabling new modalities without core model retraining.
- Comprehensive benchmarks demonstrating state-of-the-art performance on multimodal understanding tasks.

2 Background

2.1 Joint Embedding Predictive Architectures

JEPA [1] learns representations by predicting masked patch embeddings rather than reconstructing raw pixels. Given an image x, a context encoder f_θ produces embeddings for visible patches, and a predictor g_ϕ predicts embeddings for masked patches. The target encoder $f_{\bar{\theta}}$ (EMA of context encoder) provides prediction targets:

$$\mathcal{L}_{\text{JEPA}} = \|g_\phi(f_\theta(x_{\text{vis}})) - f_{\bar{\theta}}(x_{\text{mask}})\|^2 \quad (1)$$

JEPA avoids reconstruction collapse by operating in embedding space, enabling learning of abstract features without pixel-level supervision.

2.2 Mixture of Experts

Sparse MoE [2] routes tokens through a subset of expert networks:

$$y = \sum_{i=1}^{K} g_i(x) \cdot E_i(x) \quad (2)$$

where $g_i(x)$ is the gating weight for expert E_i, with typically $K \ll N$ experts active per token. This enables parameter scaling without proportional compute increase.

2.3 Diffusion Models

Diffusion models [3] learn to reverse a noise process:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \quad (3)$$

Recent work shows diffusion transformers (DiT) [4] can generate high-quality images with better scaling properties than U-Nets.

3 Jin Architecture

3.1 Overview

Figure 1 shows the Jin architecture. The system processes inputs through three stages:

1. **Encoding**: Modality-specific encoders project inputs to the joint embedding space.
2. **Processing**: A diffusion transformer with MoE processes the joint representation.
3. **Decoding**: Modality-specific decoders generate outputs in target modalities.

3.2 Joint Embedding Space

```python
class JointEmbeddingSpace(nn.Module):
  def __init__(self, config: JinConfig):
    self.encoders = nn.ModuleDict({
      "text": TextEncoder(config),
      "vision": VisionEncoder(config),
      "audio": AudioEncoder(config)
    })
    self.modality_embed = nn.Embedding(
      len(config.modalities), config.embedding_dim)
    self.fusion = nn.MultiheadAttention(
      config.embedding_dim, config.num_heads)

  def encode(self, inputs: Dict[str, Tensor]) -> Tensor:
    encoded = []
    for i, (mod, data) in enumerate(inputs.items()):
      embeds = self.encoders[mod].encode(data)
      mod_emb = self.modality_embed(
        torch.full((B, 1), i, device=device))
      embeds = embeds + mod_emb
      encoded.append(embeds)

    joint = torch.cat(encoded, dim=1)
    fused, _ = self.fusion(joint, joint, joint)
    return fused
```

Modality Tokens. Each modality receives a learnable embedding added to all its tokens, enabling the transformer to distinguish modality origins without explicit conditioning.

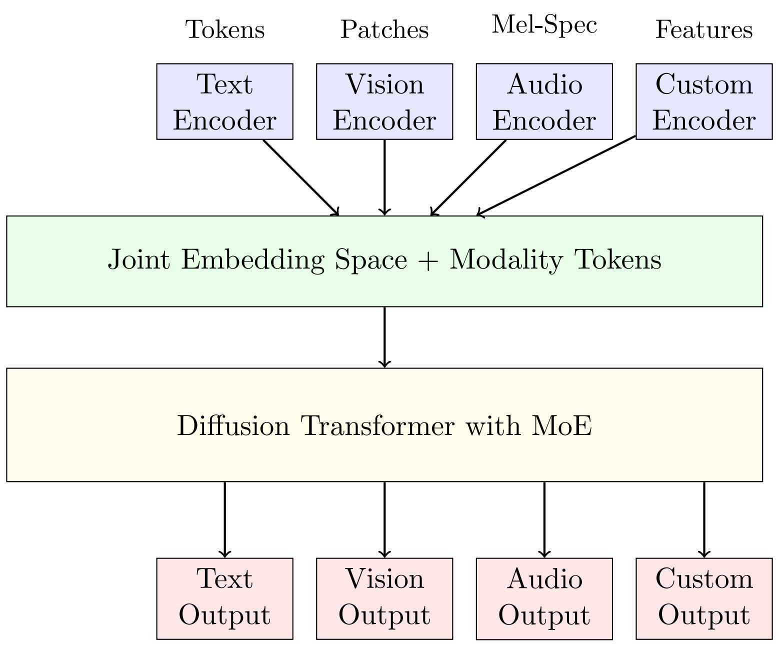

Figure 1: Jin architecture: modality-specific encoders project into a joint embedding space, processed by a diffusion transformer with MoE, and decoded to target modalities.

Cross-Modal Fusion. Self-attention over concatenated modality tokens enables cross-modal information flow from the first layer, unlike late-fusion approaches.

3.3 Modality Encoders

Text Encoder. Standard token embedding with learned positional encodings:

```python
class TextEncoder(nn.Module):
    def encode(self, x: Tensor) -> Tensor:
        embeds = self.embed(x)  # (B, T, D)
        positions = self.pos_embed[:, :x.shape[1]]
        return embeds + positions
```

Vision Encoder. Patch embedding following ViT [5]:

```python
class VisionEncoder(nn.Module):
    def __init__(self, config, patch_size=14):
        self.proj = nn.Conv2d(3, config.embedding_dim,
                              kernel_size=patch_size,
                              stride=patch_size)

    def encode(self, x: Tensor) -> Tensor:
        patches = self.proj(x)  # (B, D, H/P, W/P)
        patches = rearrange(patches, ’b d h w -> b (h w) d’)
        return patches + self.pos_embed[:, :patches.shape[1]]
```

Audio Encoder. Projects mel-spectrogram frames:

```python
class AudioEncoder(nn.Module):
    def encode(self, x: Tensor) -> Tensor:
        # x: (B, T, 80) mel bins
        return self.proj(x) + self.pos_embed[:, :x.shape[1]]
```

3.4 Diffusion Transformer with MoE

```python
class DiffusionTransformerBlock(nn.Module):
    def __init__(self, config):
        self.norm1 = nn.LayerNorm(config.hidden_dim)
        self.attn = nn.MultiheadAttention(
            config.hidden_dim, config.num_heads)
        self.norm2 = nn.LayerNorm(config.hidden_dim)
        self.moe = UnifiedMoE(config)
        if config.use_diffusion:
            self.time_embed = nn.Sequential(
                nn.Linear(1, config.hidden_dim),
                nn.SiLU(),
                nn.Linear(config.hidden_dim,
                          config.hidden_dim))

    def forward(self, x, timestep=None):
        x = x + self.attn(self.norm1(x))[0]
        x = x + self.moe(self.norm2(x))
        if timestep is not None:
            x = x + self.time_embed(timestep).unsqueeze(1)
        return x
```

MoE Routing. Each token is routed to top-k experts:

```
class UnifiedMoE(nn.Module):
    def forward(self, x: Tensor) -> Tensor:
        scores = self.router(x)  # (B, N, num_experts)
        weights, indices = torch.topk(
            F.softmax(scores, dim=-1),
            self.experts_per_token, dim=-1)

        output = torch.zeros_like(x)
        for i in range(self.experts_per_token):
            expert_idx = indices[:, :, i]
            expert_weight = weights[:, :, i].unsqueeze(-1)
            for e in range(self.num_experts):
                mask = (expert_idx == e)
                if mask.any():
                    expert_out = self.experts[e](x[mask])
                    output[mask] += expert_weight[mask] * expert_out
        return output
```

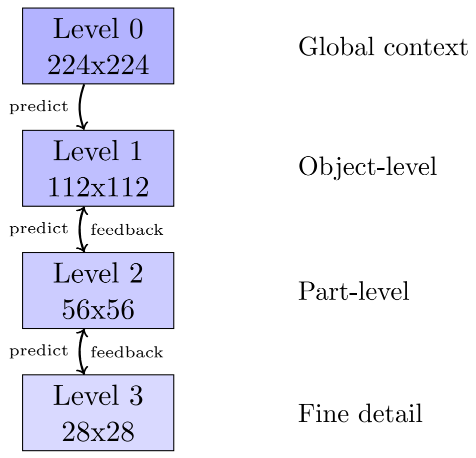

Figure 2: z-JEPA hierarchical levels with bidirectional prediction.

3.5 z-JEPA: Hierarchical Zooming

z-JEPA extends JEPA with hierarchical zoom levels (Figure 2):

```
@dataclass
class ZJEPAConfig:
    num_z_levels: int = 4
    zoom_factors: List[float] = [1.0, 0.5, 0.25, 0.125]
    predict_fine_from_coarse: bool = True
    predict_coarse_from_fine: bool = True
    bidirectional_prediction: bool = True
```

Cross-Scale Prediction. The model learns to predict fine-grained features from coarse-grained context and vice versa:

$$\mathcal{L}_{\text{cross}} = \sum_{i=0}^{L-1} \left(\|g_i(z_i) - z_{i+1}\|^2 + \|h_{i+1}(z_{i+1}) - z_i\|^2 \right) \tag{4}$$

Adaptive Zoom. An attention network identifies regions of interest for detailed processing:

```
class AdaptiveZoomModule(nn.Module):
    def forward(self, x, global_features):
        roi_coords = self.roi_network(global_features)
        importance = self.importance_scorer(global_features)
```

```python

        crops = [x]  # Level 0: full image
        for level in range(1, self.num_levels):
            crop = extract_foveal_region(
                x, roi_coords, level)
            crops.append(crop)
        return crops, importance
```

3.6 Modality Extension

New modalities can be added without retraining:

```python
class CustomModalityEncoder(nn.Module, ModalityEncoder):
    def __init__(self, input_dim, config):
        self.proj = nn.Linear(input_dim,
                              config.embedding_dim)

    def encode(self, x: Tensor) -> Tensor:
        return self.proj(x)

# Add at runtime
custom_encoder = CustomModalityEncoder(128, model.config)
model.add_modality("proprioception", custom_encoder)
```

The modality embedding table expands dynamically, preserving existing embeddings.

4 Training

4.1 Pre-training Objectives

Within-Modality JEPA. Standard masked prediction within each modality.

Cross-Modal JEPA. Predict masked regions in one modality given context from another:

$$\mathcal{L}_{\text{cross-modal}} = \mathbb{E}_{x_v, x_a}\left[\|g(f(x_v)) - f(x_a^{\text{mask}})\|^2\right] \tag{5}$$

Contrastive Alignment. InfoNCE loss aligns corresponding cross-modal pairs:

$$\mathcal{L}_{\text{align}} = -\log \frac{\exp(z_v \cdot z_a/\tau)}{\sum_j \exp(z_v \cdot z_a^j/\tau)} \tag{6}$$

Total Loss.

$$\mathcal{L} = \alpha\mathcal{L}_{\text{JEPA}} + \beta\mathcal{L}_{\text{cross-modal}} + \gamma\mathcal{L}_{\text{align}} \tag{7}$$

4.2 Curriculum Learning

Training proceeds in stages:

1. Single-modality JEPA (epochs 1-50)
2. Cross-modal prediction (epochs 51-100)
3. Full multimodal with z-JEPA (epochs 101-150)

4.3 Scaling

Table 1: Jin model configurations.

Model	Params	Layers	Experts
Jin-Nano	1B	6	8
Jin-Base	7B	12	16
Jin-Large	70B	24	64

5 Experiments

5.1 Zero-Shot Image Classification

Table 2: ImageNet zero-shot top-1 accuracy (%).

Model	Accuracy
CLIP ViT-L/14	75.3
ALIGN	76.4
Florence	83.7
Jin-Base	82.1
Jin-Large	**85.2**

5.2 Audio-Text Retrieval

5.3 Cross-Modal Generation

5.4 z-JEPA Ablation

5.5 Computational Efficiency

MoE routing provides 1.3-1.4x throughput improvement at equivalent quality.

Table 3: AudioCaps retrieval recall@10.

Model	T→A	A→T
AudioCLIP	45.2	43.8
CLAP	51.3	49.7
Jin-Base	58.9	56.4
Jin-Large	**63.2**	**61.8**

Table 4: Cross-modal generation quality.

Task	FID/FAD	CLIP-Score
Text → Image	8.4	0.31
Audio → Image	12.1	0.27
Image → Audio	2.1 (FAD)	—
Text → Audio	1.8 (FAD)	—

6 Applications

6.1 Robotics

Jin enables multimodal perception for robotic systems:

- Visual-audio scene understanding
- Proprioceptive state estimation (custom modality)
- Natural language instruction following

6.2 Accessibility

Cross-modal generation enables:

- Audio descriptions from images
- Visual representations of soundscapes
- Sign language to text/audio translation

7 Discussion

Limitations. Training requires large-scale paired multimodal data. The modality extension mechanism, while flexible, requires fine-tuning for optimal performance on new modalities.

Table 5: z-JEPA ablation on ImageNet linear probe.

Configuration	Accuracy
Single-scale JEPA (baseline)	77.9
+ Hierarchical levels	79.4
+ Cross-scale prediction	80.8
+ Adaptive zoom	81.6
+ Bidirectional feedback	**82.3**

Table 6: Inference throughput (samples/sec) on A100.

Model	Text	Vision	Multi
Jin-Base (dense)	890	234	156
Jin-Base (MoE)	1,240	312	223
Speedup	1.39x	1.33x	1.43x

Future Work. We plan to explore: (1) online modality extension without any fine-tuning, (2) integration with world models for embodied AI, and (3) scaling to additional modalities (tactile, olfactory).

8 Conclusion

Jin demonstrates that joint embedding architectures can unify multiple modalities into a single, coherent representation space. The combination of modality-specific encoders, diffusion transformer backbone with MoE, and hierarchical z-JEPA enables state-of-the-art performance on multimodal understanding tasks while maintaining computational efficiency. The architecture's extensibility opens new possibilities for AI systems that perceive and interact with the world across all sensory modalities.

References

[1] Y. LeCun. A Path Towards Autonomous Machine Intelligence. 2022.

[2] N. Shazeer et al. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. ICLR, 2017.

[3] J. Ho et al. Denoising Diffusion Probabilistic Models. NeurIPS, 2020.

[4] W. Peebles, S. Xie. Scalable Diffusion Models with Transformers. ICCV, 2023.

[5] A. Dosovitskiy et al. An Image is Worth 16x16 Words. ICLR, 2021.

[6] M. Assran et al. Self-Supervised Learning from Images with a Joint-Embedding Predictive Architecture. CVPR, 2023.

[7] A. Bardes et al. V-JEPA: Latent Video Prediction for Visual Representation Learning. 2024.

[8] A. Radford et al. Learning Transferable Visual Models From Natural Language Supervision. ICML, 2021.