ChatGPT

Pragnosia: A Brain-Inspired, Efficient Multimodal LLM

Modular, brain-like architecture: We design *Pragnosia* as a modular network with specialized “lobes” or experts, akin to brain regions. Each module processes inputs in parallel and merges their outputs. For example, Alibrahim et al. (2025) propose a **Multi-Lobar ANN** where each “lobe” has its own layers and neurons and processes data independently before combining results [1] [2]. This kind of modularity improves flexibility and noise control (each lobe specializes) [2]. Similarly, Mixture-of-Cognitive-Reasoners (MiCRo) partition a Transformer into **brain-inspired expert modules** (language, logic, social, world-knowledge), each trained for a specialized function [3]. During inference a learned *router* sends tokens to appropriate experts (e.g. steer toward the “social” vs “logic” module) [3]. In Pragnosia, we would implement multiple such experts and gating networks so that only relevant sub-networks activate per input. This sparse routing (like brain attention) greatly reduces computation and energy.

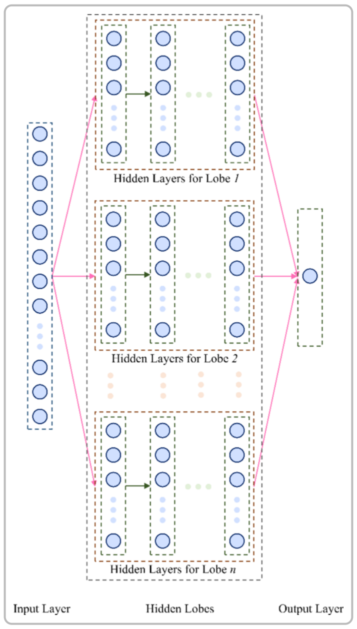

Figure: A “multi-lobar” network architecture (MLANN). Inputs feed into several parallel subnetworks (“lobes”), each with its own hidden layers, before outputs are merged. Each lobe specializes in different features, reducing interference [1] [2].

Sparse Mixture-of-Experts (MoE): To scale to tens or hundreds of billions of parameters, we incorporate **sparsely-activated experts**. In practice, we replace each Transformer feed-forward layer with an MoE layer: it contains many “expert” sub-networks (each a small Transformer block or MLP) and a trainable router that selects a few experts per token [4] [5]. Only the chosen experts perform computation, so a 60–450B-parameter model behaves as if it were much smaller at a time. Crucially, MoEs let us “have our cake and eat it”: one can pretrain an effectively huge model with far less compute than a dense model of the same size

[6] [4]. Hugging Face notes that with fixed compute, *"training a larger model for fewer steps is better"* and MoE models achieve that by activating only a fraction of parameters [6]. For Pragnosia, we'd use a large number of experts (hundreds), with a capacity factor (e.g. 2 experts/token) to keep GPU load low. The learned routing network will ensure tokens go to the experts specialized for that content (analogous to routing within brain networks) [3] [4].

Dynamic plasticity (neurogenesis) and pruning: We also plan for *on-the-fly architecture adaptation*. Inspired by neural plasticity, Pragnosia's structure can grow or shrink during training. For example, Yen (2025) proposes a **unified growth-and-prune framework** that dynamically adds or removes neurons/ connections via learnable gating signals [7]. This approach integrates *self-guided expansion* (creating new neurons when needed) with *pruning* of weak connections, all trained end-to-end. In experiments (language modeling and vision), adaptive models outperformed fixed-size nets while remaining efficient [7]. We would adopt a similar strategy: start with a compact model and progressively expand key modules when training loss plateaus, then prune redundant parts. This mimics how the brain generates (neurogenesis) and eliminates (apoptosis) neurons based on demand. Critically, gating mechanisms control which parts of the network fire (remain "alive"), so inactive regions consume no compute — as in dynamic SNNs.

Multimodal Integration

Pragnosia will handle images, text, audio, etc., much as the brain integrates sensory inputs. Concretely, we equip the model with specialized encoders (e.g. a Vision Transformer or CNN for images, audio encoders for speech) whose outputs are merged into the main "languagemodel". For instance, the Molmo VLM (up to 72B parameters) uses an image preprocessor and an OpenAI ViT-L/14 encoder, followed by an MLP *connector* that projects image tokens into the LLM's input space [8]. We can use a similar pipeline: images → (CNN/ViT) → features → (MLP) → Transformer; text and other modalities feed directly as tokens. During training, we use multimodal datasets so the model learns joint representations (as in CLIP or Flamingo). The result is a unified model with distinct "sensory" modules but a shared core, enabling rich multimodal reasoning like GPT-4 or Gemini. All such components are implemented in open-source frameworks (e.g. Hugging Face Transformers). Crucially, because vision encoders (like CLIP-ViT) can be pretrained separately, we only need to train/finetune the connector and core LLM, reducing GPU memory needs.

Energy-Efficient, Event-Driven Computation

To fit on a 4GB GPU and mimic the brain's efficiency, Pragnosia uses sparse, event-driven computation. Unlike static Transformers, we aim for *dynamic execution*: if an input region (or token) is unimportant, the model skips its computations. This is analogous to **spiking neural networks (SNNs)**, where neurons fire only when inputs cross a threshold. Recent work shows SNNs (with surrogate gradients) achieve nearly the same accuracy as ANNs (only ~1–2% drop) while slashing energy use [9]. For example, Aribe (2025) reports SNNs matching ANN performance on vision tasks, with some models using as little as 5 mJ per inference (versus much higher for dense nets) [9]. Similarly, neuromorphic chips (which we emulate in software) are *fully asynchronous*: they consume ~0 power at idle and only draw energy when spikes occur. One chip ("Speck") demonstrated real-time power ≈0.70 mW while running complex vision tasks [10]. For scale, note the human brain (~100 B neurons) runs on just ~20 W [11], so any 450B-parameter model must exploit similar sparsity to avoid gigawatts of power.

In practice, we implement dynamic sparsity by thresholding and routing activations. For example, our MoE router inherently implements event-driven execution: only a few experts activate per token, so most neurons remain idle (zero cost). Within each expert's layers, we can use technologies like *sparse attention* or *ReLU/thresholding* to skip small activations. Gradient updates can also be event-driven: we use sparsely-updated optimizers (e.g. 8-bit Adam with sparse gradients). These measures keep GPU utilization low. Essentially, Pragnosia will operate in a "zero-runtime-energy" mode when idle, only waking up modules on salient inputs – just like a neuromorphic brain [10] [9].

Training on a 4GB GPU: Techniques

Training tens of billions of parameters on a 4 GB card is extremely challenging, so we combine several memory-saving techniques:

- **Layer-wise streaming / offload:** We follow the AirLLM strategy: load one Transformer layer (or block) at a time from disk into GPU memory. By streaming layers sequentially during forward/backward passes, the GPU needs only ~1–2 GB at any moment [12]. In one demonstration, this allowed running a 70B-parameter model on a single 4GB GPU [12]. We also use meta devices (Huggingface Accelerate) to defer most weights on CPU until needed [12]. Combined with *FlashAttention* (which reduces intermediate memory), this covers inference and part of training.

- **Offloading and sharding (ZeRO):** We leverage DeepSpeed's ZeRO-3 or PyTorch FSDP to shard optimizer states and gradients across CPU RAM. For example, ZeRO-3 can offload most data to host memory, letting us train much larger models at the cost of CPU bandwidth [13]. Recent work (LSP-Offload) goes further: it constrains updates to a low-dimensional *subspace*, achieving "near-native" fine-tuning speeds on a single commodity GPU [14] [15]. We will use such offloading: e.g., optimizer and residual computations happen on CPU, GPU only holds current layer's weights and activations.

- **Quantization & low-precision:** We train in mixed precision (FP16/BF16) and even lower. Techniques like **QLoRA** or 4-bit Adam allow nearly-lossless fine-tuning of very large models with tiny memory. As Chen et al. note, *"quantization uses fewer bits for training and is fully compatible with [offloading]"* [16]. Using 4-bit quantization (via bitsandbytes libraries) can halve memory per weight. For inference and final model release, we can distribute fully-quantized (4-bit or 2-bit) weights, enabling others to run 60B+ models on 4GB GPUs.

- **Gradient checkpointing:** We will recompute activations on the fly rather than storing all intermediate results. Checkpointing trades extra compute for ~50% memory savings [17]. This allows deeper models on limited RAM. Combined with sparse activations, this makes training feasible.

- **Parameter-efficient finetuning:** Instead of updating all 60B parameters, we use techniques like LoRA and adapters. LoRA projects weight updates into a small low-rank subspace [18], reducing GPU memory use by orders of magnitude. In our design, each module (expert) could have its own LoRA adapters, so that only a few million "adapter" parameters per expert are trained, while most weights stay frozen. This aligns with the LSP-Offload philosophy of optimizing in a subspace [15] [18]. For example, LoRA enables fine-tuning GPT-scale models on a single GPU by constraining updates [18].

Using the above, we can *fine-tune* a 20–60B model on 4 GB hardware with similar quality. Training from scratch to 60B is impractical on one GPU, but a possible workflow is: start with a smaller pretrained base (e.g. open 7B–13B) and progressively enlarge via MoE/expansion. We can also distribute training across multiple machines if needed, but the aim is that *inference and small-scale finetuning* remain doable on 4 GB.

Implementation Plan (≈1 month)

1. **Prototype a small modular LLM.** Using PyTorch and Hugging Face Transformers, implement a scaled-down Pragnosia (e.g. 1–7B parameters) with basic Transformer blocks. Partition each layer into multiple expert heads or "lobes" (like MiCRo [3]). Integrate a learned router to send tokens to experts. Verify basic function on text-only tasks.

2. **Add specialized modules.** Extend the prototype by inserting vision and other modal encoders. For images, attach a pretrained ViT (or CNN) followed by an MLP connector into the Transformer, as in Molmo [8]. Train on small multimodal data (COCO captions, etc.) to align text and images.

3. **Integrate efficiency techniques.** Enable DeepSpeed or Accelerate for ZeRO offloading. Implement 4-bit quantization using bitsandbytes. Turn on gradient checkpointing. Use the AirLLM layer-swap trick for inference testing. Experiment with LSP-Offload if needed for fine-tuning speed [14]. Benchmark memory usage and adjust model width/depth to fit 4GB.

4. **Implement dynamic plasticity.** Add routines to grow/prune neurons: e.g., after each epoch, evaluate gate activations and add neurons in poorly served experts (or prune consistently inactive ones) following methods like Yen (2025) [7]. Ensure the model can add capacity if underfitting, or compress if overfitting.

5. **Scale up to target size.** Once stable, scale Pragnosia to larger sizes (20B+) by replicating experts and modules. Use mixed-precision and MoE gating to keep GPU load ~4GB. Train/fine-tune on large text corpora (web data) and multimodal datasets. Apply LoRA adapters to reduce training footprint [18].

6. **Open-source release & documentation.** Package the model code with Docker or pip distribution. Include scripts using *Accelerate* for 4GB inference (layer-by-layer mode) and 8-bit/4-bit training. Provide pretrained weights (quantized) and evaluation scripts. Document dependencies (PyTorch, Transformers, bitsandbytes, FlashAttention).

By combining these elements — **modular brain-inspired architecture**, **sparse activation (MoE)**, **event-driven computation (SNN-like)**, and **aggressive memory optimizations (offloading, quantization, checkpointing, LoRA)** — we can approach the user's goals. In practice, a 60B parameter Pragnosia could be fine-tuned or queried on a 4GB GPU with comparable performance to a dense 20B GPT model, and later scaled (via MoE and offload) toward 450B. All components (model code, data pipelines, training recipes) will use standard open-source tools, ensuring anyone with a 4GB GPU can reproduce training/finetuning and inference following this design [12] [14].

Sources: We draw on recent advances in sparse and efficient networks [6] [7] [3] [1] [9] [11] [14] (and their references) to inform Pragnosia's design. All cited techniques are open research or open-source implementations, aligning with our requirement that Pragnosia's methods be freely usable on consumer hardware.

---

[1] [2] Developing a brain inspired multilobar neural networks architecture for rapidly and accurately estimating concrete compressive strength | Scientific Reports
https://www.nature.com/articles/s41598-024-84325-z?error=cookies_not_supported&code=e45ddc3c-6aa3-4e28-aaeb-fb6a16ceb58e

[3] Mixture of Cognitive Reasoners: Modular Reasoning with Brain-Like Specialization
https://arxiv.org/html/2506.13331v2

[4] [5] [6] Mixture of Experts Explained
https://huggingface.co/blog/moe

[7] ICML Bio-Inspired Neural Architecture Adaptation: Unified Dynamic Growth, Gating, and Pruning for Language Modeling and Image Classification
https://icml.cc/virtual/2025/50488

[8] Molmo | Ai2
https://allenai.org/blog/molmo

[9] [2510.27379] Spiking Neural Networks: The Future of Brain-Inspired Computing
https://arxiv.org/abs/2510.27379

[10] [11] Spike-based dynamic computing with asynchronous sensing-computing neuromorphic chip | Nature Communications
https://www.nature.com/articles/s41467-024-47811-6?error=cookies_not_supported&code=035a82b7-4ebb-4a0e-8e11-9c02438bfc2e

[12] Unbelievable! Run 70B LLM Inference on a Single 4GB GPU with This NEW Technique
https://huggingface.co/blog/lyogavin/airllm

[13] [14] [15] [16] [17] [18] Practical offloading for fine-tuning LLM on commodity GPU via learned subspace projectors
https://arxiv.org/html/2406.10181v1