EvoNet: Towards Self-Evolving, Entropy-Guided AI

Leonhard Waibl

July 2025

1 Introduction - Growing Minds, Not Just Bigger Models

1.1 TL;DR:

This post introduces EvoNet, a proposal for a fundamentally new type of neural network. Instead of endlessly scaling rigid models by cramming in more parameters, I want to explore a system that can grow and restructure itself. This is achieved by maintaining a persistent internal state during operation, much like our brain. The architecture is designed to self-expand and self-supervise, driven by entropy and information density, while still running on conventional GPUs.

In this post, I outline the motivation behind this idea, the rough design of the architecture, some implementation details, and why this approach might lead to an entirely new path beyond today's brute-force scaling.

Contents

1.2 The Current State of Rigid Models

Today's neural networks act like frozen machines. They have both a fixed architecture and a fixed computation path. For example, in transformer models, each token is computed in the same way. Meaning computation is deterministic and uniform, regardless of the input's complexity.

There are techniques used to introduce **variation and randomness**, like sampling temperature. However, these are external modifications, not intrinsically architectural features.

Additionally, models aren't capable of pausing, reflecting, or modulating their computational depth in response to the task's difficulty. While modern models are capable of reasoning through "**chain-of-thought**" on paper, this also isn't an intrinsic capability. Typically, an external loop reprompts the model multiple times, trying to elicit a better answer. However, this loop has no sense of uncertainty and also can't alter the model's internal computation patterns.

I've already worked on this issue in another project titled Adversarial Alignment, which aims to predict model uncertainty from latent knowledge. There are multiple applications ranging from alignment to interpretability. However, the most significant advantage lies in production. There, it could enable dynamic reasoning depth based on uncertainty. That is something current models don't support.

1.3 How Model Rigidity Limits Intelligence

In the previous paragraph, I discussed the architectural limitations of state-of-the-art (**SOTA**) models. These directly translate into limitations in thinking and reasoning ability.

The most significant problems arise in situations where a task doesn't align with a model's capabilities or computation structure, defined during training. In these situations, the model is unable to determine the correct strategy, and the output becomes unpredictable, leading to overconfident but incorrect outputs, often referred to as "**hallucinations**." In such situations, the model adheres to the notion of merely writing a plausible text, regardless of its accuracy. Similar to a student trying to gain some points in an exam by writing something incorrect but plausible to earn additional points.

Another drawback of current models is that, in each iteration, all the already elaborated concepts get discarded at the end of every forward pass, which means internal attention activations can't be saved and must be recomputed for every token.

A result of this is that models are required to process all the information in a single pass and on every pass. This behavior severely limits their efficiency, as every output, regardless of significance, gets the same amount of resources.

Restricting all processing to a single forward pass also requires dramatically larger model sizes to tackle all problems, as a model has to be prepared for anything on this single pass.

This is also why we see modern models with hundreds of billions of parameters. It is not because every problem inherently requires that scale, but because the architecture has no alternative way to handle complex reasoning without brute-force redundancy.

However, techniques such as Mixture-of-Experts (**MoE**) exist, enabling dynamic, context-dependent computation paths that vary according to the task. These reduce the required computation significantly, but problems such as discarded memory or compute redundancy still persist.

Now I will also go into other issues we encounter in SOTA models. These are rooted in the data flow through the model. Data generation occurs in a single forward pass, in which information might encounter different bottlenecks. For example, a single multilayer perceptron (MLP) layer has limited expressivity, thereby limiting the amount of information that can be passed through the network.

Additionally, model structures today are fixed at the time of training. While it is possible to fine-tune the weights for a new task, there is no mechanism to expand or restructure the model after it has been trained.

If a model proves insufficient for increasing complexity, a completely new model has to be trained, as knowledge from one model cannot be directly transferred to another.

Also, minor architectural changes necessitate retraining the entire model, even though techniques such as Alpaca (transfer learning) or LoRA (partial retraining) can mitigate this issue.

1.4 How Biological Brains Are Different

In contrast, biological brains are dynamically adaptive systems.

Real neurons don't just vary their synaptic strength, like weights in artificial neural networks (**ANNs**). They also physically grow, rewire, and restructure. This dynamic flexibility allows brains to solve problems not just by tweaking parameters, but by fundamentally changing the information flow. This plasticity lets them self-organize and continually adapt their structure to new tasks or environments.

Another significant advantage of biological systems is that they retain a persistent internal state. Information isn't just passed forward; it is also stored in the network. This behavior allows for working memory, attention span, and context sensitivity. All features that current artificial networks lack. Biological systems also tightly regulate their overall activity, avoiding both runaway excitation and total quiescence. This balance is critical for stable computation.

The closest artificial counterparts are spiking neural networks (**SNNs**), which simulate neuron-like behavior, including event-based processing and state retention. They are more energy-efficient and, in some cases, offer better generalization than conventional ANNs. But there are three huge drawbacks. They are challenging to train and scale. As SNNs also work in the time domain, their input and output formats are often incompatible with standard ML pipelines. And at last, they either require special hardware to run or are processed on the CPU.

All these factors combined severely limit the practical deployment of such networks on today's GPU-centered infrastructure.

1.5 What a Better System Might Look Like

The idea I propose here is a neural architecture that can both self-improve and self-expand, while also running on conventional GPU infrastructure.

This section provides a brief overview of the proposed architecture. More details are discussed in later sections, specifically on implementation. Some implementation details, particularly those related to training and self-alignment, were intentionally omitted, as they would significantly lengthen the post but are not strictly necessary for understanding the basic architecture. These details will be shared in later posts.

The proposed architecture fundamentally changes how artificial networks handle data. Instead of a shallow, one-shot pass, the network evolves activations over time within a living structure. The model maintains a persistent state, similar

to an SNN or our brain. Information flowing through the network is also time-dependent and not processed as a simple, straight-through pass. The difference in computation flow to conventional ANNs can be best understood with an analogy to railway networks, which also highlights one of the main weaknesses of applying this new architecture in production.

Imagine there is a factory that produces what we need. In the example of conventional LLMs, the next token. To get our total output, we need to send a train with the whole context window to the factory. It puts one additional token onto the train and sends it back. Then we check this token and send the train again, now with this token added. And so on.

The network is simply a single rail that connects the user to the factory, with no complicated networks.

This is different from the proposed architecture. Here, there are multiple factories, each specializing in a different task. The input is loaded onto multiple trains, which carry the information to the various factories and also connect them. The advantage is that we can process everything in one go, and the network allows data to flow efficiently. We only need to send the information once and receive the output.

However, here we encounter the biggest drawback of the self-expanding architecture. Let the rail network now represent the individuality of data processing. In the first example, the network is easy to build because it is just a single straight line. This simplicity makes it trivial to parallelize and queue requests, as every user gets treated the same, and it does not matter what happened before, since everything gets processed in one go. The other approach is different. Here, the railway network is complex and requires individual ones for each user to prevent intermixing with other data. This constraint makes the application difficult. For every user request, the network would need to be reloaded.

In production, this would result in a significant personal network footprint. Additionally, a lot of bandwidth would be needed to save and load these personalized models.

The above part described the differences in application and also the main drawback of this architecture.

However, the main benefit would not be in such widespread applications, but in enabling the AI to respond and act more like a human entity. This behavior stands in stark contrast to current models, which handle complexity almost exclusively by scaling up parameters rather than structurally adapting to them.

Also reflecting on efficiency and scale. It becomes clear why scaling a single all-purpose factory is far more demanding than coordinating multiple specialized ones.

Here are the key factors that distinguish the newly proposed architecture from SNNs, and how the key advantages, such as self-supervision, self-expansion, and GPU compatibility, will be achieved.

First, to self-supervision and entropy-based learning. The general idea here is that entropy is a measure of uncertainty, and reducing it would mean making the network less uncertain, in other words, making it learn. It is akin to bringing order to a chaotic system. The entropy of the system, estimated during runtime, is then used to change weights. How this exactly occurs will be discussed later. It is also not finalized yet, as I need to run tests to determine the best implementation of this behavior. There are also other measures for learning and alignment, which, when initialized, should self-supervise the system. This topic is briefly discussed at the end and will be covered in more detail in a subsequent post.

Another idea is to utilize information density to expand the underlying network automatically. When the network gets oversaturated with too much information, when entropy remains high, or when nodes get over-activated, the network expands. Depending on the type of overuse, the network either grows a new node, connection, or cluster. Also to be discussed in another post. This behavior is also similar to how our brain evolves in childhood and adolescence. It enables specialization, efficiency, and structural adaptation.

GPU compatibility is achieved by iteratively updating the network, rather than computing spikes independently, as is done in classical SNNs. This design decision enables us to compute the entire network in a single large, sparse matrix multiplication. To optionally allow for different timings in the future, a signal delay might also be implemented.

In the future, I plan to show concrete examples of these mechanisms in small prototype systems.

2 Exploring This New Kind of System

I have already laid out the benefits and core concepts of this new architecture.

Here, I'll delve deeper into the implementation details.

Finally, I'll also discuss the current hurdles that are preventing me from making further progress on the project.

Let's begin by looking at the information flow through the model and its concept of memory.

2.1 Persistent Model State

This architecture features a persistent model state, saving information in the model's compute path. This is enabled by a bidirectional and circular data flow. This method is fundamentally different from current architectures, where data flows in a single direction and the network resets after each pass.

In the architecture, information remains within the model, organically absorbing new inputs and retaining them within the network. This persistent state enables the network to refine its representations over time, rather than recomputing everything from scratch for each input.

Here we can see two diagrams showing the computation paths, the arrows represent connections (green and red represent weights):

First, in conventional ANNs, the data flows only forward and produces an output.

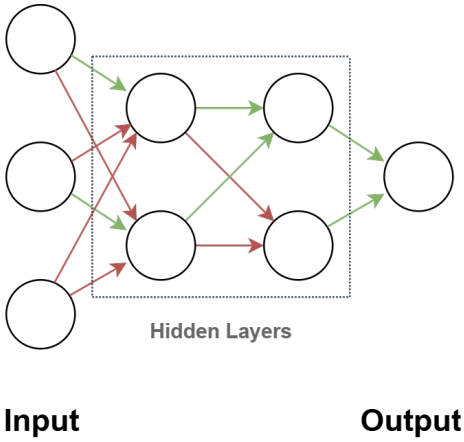

Figure 1: Conventional ANN

In the second image, we see the self-expanding network. There, the arrows do not all point in the same direction, and also route backward. This structure causes a persistent state in the model.

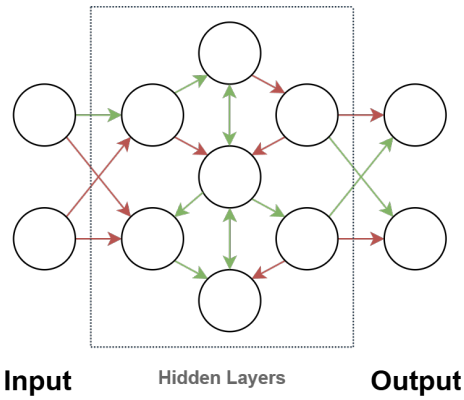

Figure 2: EvoNet

The network is also better described as a graph, rather than in terms of layers. It is a structure that can naturally route information in complex ways, unconstrained by a simple pipeline. This could significantly increase the computing capability, as connections and nodes are not limited by any dimension. This freedom could also enable networks of four, five, or higher dimensions. This dimensional freedom is also one advantage over our human brains, which are constrained to 3 spatial dimensions. More dimensions could allow for even better reasoning.

The biggest problem here is getting the network initially started, as processing assumes the network already holds a stable passive state to build upon. I will also discuss other issues related to this persistent state in the subsequent parts. Computation is discussed further in the chapter on GPU compatibility.

2.2 Entropy and Energy - Training and Stabilisation of the Model

Information theory is used extensively in this architecture. Entropy plays a dual role here. It maintains the network's dynamic stability and drives learning by guiding the adjustments to weights. We also have two concepts related to uncertainty: entropy and energy.

In this architecture, entropy measures uncertainty, while energy reflects the overall activation present in the system. Entropy expresses uncertainties, both aleatoric and epistemic.

Aleatoric uncertainty is exclusively used for training. The brief explanation of how this training process works is provided at the end of this section. Epistemic uncertainty will be used later to estimate information density and expand the network.

When referring to energy, I also mean a kind of uncertainty, but seen as the sum of the activations in the system. It is similar to thermal energy in physics, but here it refers to the activations. So, general variance across activations in the network.

This concept is used to stabilize the network by maintaining a small positive total network energy sum. This keeps the network in balance even through self-feedback.

It is a valid concern that, by feeding back into itself, the network could drift into a state of maximum variability, where node values become very large, and the network only acts as a spike generator.

Generally, this means activations tend toward one direction. For example, the

total energy approaches infinity. All signals converge to nearly the same activation level, effectively erasing meaningful distinctions. The network becomes a uniform sea of activity, unable to compute or represent anything.

At the other end lies the zero state, where all network nodes fall into the inactive state, with no activation.

It is similar to what you learned in elementary physics or a control class. Imagine it like balancing a pendulum. The energy in the system describes the deviation of the position of the pendulum from equilibrium. If we have too much energy, the pendulum becomes unstable, and we likely overcorrect, causing it to fall down the other side. This is also why it is so hard to balance an umbrella with your hand. Absolute equilibrium is also undesirable, as it implies the system is perfectly balanced. It would theoretically be the desired state, but we can't tell which equilibrium it is. So we could have landed in the zero states.

For expansion and training, this variation is also necessary, as it provides us with indications of where the network will need to be extended. In the next section, I will discuss this further.

Additionally, input energy must be taken into consideration to prevent the introduction of any additional energy that could destabilize the system.

For training, aleatoric uncertainty from the output is used to train the model. The goal is to minimize this uncertainty, enhancing model expressiveness and reducing variance. The training process occurs similarly to a real neural network, strengthening connections that lead to the correct output, which is also evaluated by signal strength. Otherwise, there is also a weight and activation decay that should discourage the network from over-relying on certain connections. In future posts, I will explore how this carefully balanced dynamic allows the network to both remain stable and adapt its structure. At the end, there is also a section on alignment and learning, which addresses other forms of uncertainty and oversight. And how this system should be capable of self-evaluation.

2.3 Expansion Through Information Density

This architecture also supports dynamic expansion. This is enabled by estimating the information density at different points in the network through the epistemic uncertainty of nodes. This is calculated, as mentioned above, by the activation of the node. More precisely, by the variance of its inputs. Contradictory inputs with strong activation hint at an information overflow at this point. Instead of simply increasing all weights or adding more generic capacity everywhere, this approach grows new nodes exactly where the information load demands it. A similar phenomenon occurs with information overload in conventional ANNs, but it manifests in the gradients.

Depending on this, a new node is added to this part of the network with small initial weights. Through training, this should reduce the information load on the single node, allowing for better processing. It also enables the network to adapt to more complex tasks over time. Local expansion handles concentrated information pressure, while cluster-level growth allows the network to diversify and tackle entirely new functional domains.

Cluster expansion is another method by which the network can grow. A new cluster is inserted into the network and connected to existing ones. The main idea is that the stability of the existing clusters should stabilize the new cluster. These are comparable to brain regions and should enable the network to expand its skills, acting somewhat like Mixture-of-Experts (MoE) in conventional ANNs to solve various problems. Just as biological brains not only grow new connections but also prune unused ones, future versions of this system could incorporate mechanisms to remove or downscale underutilized nodes. This also plays a massive part in the concept of self-alignment for the network. But that concept will be covered in detail in a future post. The description of the fundamental self-alignment idea can be found at the bottom of the post.

In this way, the network does not just passively process inputs. It actively reorganizes itself, evolving a richer internal structure tailored to its learning history.

2.4 GPU-Friendly Computation

One of the significant drawbacks of the most closely linked spiking neural network (SNN) architecture is that it either requires specialized hardware to run or is sequentially processed on a CPU, making it incompatible with the current AI infrastructure, which heavily relies on GPUs.

The proposed architecture solves this compatibility problem by computing the network iteratively. In each iteration, every node passes its current state to the next. This means the whole network can be computed at once as a single sparse matrix operation. Unlike traditional SNNs, which simulate spikes step by step, this design updates everything simultaneously.

Sparse matrices also skip over zero entries, so the GPU only processes active connections, making it very efficient by calculating only the required paths. Especially compared to typical dense networks, which process every weight, regardless of whether it contributes meaningful computation.

The cluster-like structure also facilitates model parallelism. The model can be split into clusters across different GPUs. These clusters then handle their data internally on the GPU with high bandwidth. Only the links between clusters need to be shared across GPUs, which significantly reduces the inter-GPU communication bandwidth required. This is similar to how different brain regions process information independently but still communicate through longer pathways.

This setup should work with today's GPU systems and can also scale up as GPU interconnects and memory get better.

3 Path to Implementation

Currently, there is no working prototype. There are two primary reasons: funding and complexity.

This is a personal, self-funded research effort, and I am operating with minimal computing resources and no institutional backing.

The other point is the scope of the project itself, which became very complex relatively quickly.

I have already written over 1000 lines of original code, since AI assistants like GitHub Copilot or ChatGPT cannot understand what the project should do, at least a year ago. This highlights just how novel the project is. It is outside the realm of what typical ML code completions or pre-trained examples can even recognize.

Making development a lot more time-intensive. The existing codebase already implements core structures, such as nodes, clusters, and the iterative state-passing mechanism. This already took me over two weeks, as I also needed to develop the algorithms and structure.

As a result, I focused on another project, titled Adversarial Alignment, for some time. It applied the alignment paradigms developed for this architecture to current LLMs. There, I also encountered funding issues, as I cannot afford to test it on large enough models to obtain meaningful results.

I plan to continue this project by proving the model's learning ability with a game like DOOM. A game like DOOM is ideal for early testing because it offers a controlled environment with complex yet observable patterns, making it perfect for proving adaptive learning. Demonstrating learning on DOOM would be a clear and intuitive demonstration of the system's ability to adapt and improve in a dynamic environment.

It is rare for such fundamental architectural experiments to be pursued independently, outside of well-funded labs, which makes finding ways to continue all the more critical. Anyone interested in collaborating on this or contributing compute is very welcome to reach out. Refer to the Call for Collaboration at the bottom of the post.

Next, I will outline the current stage of progress and the planned development path to show where this project is headed. The Self-Regulation and Alignment part is especially interesting, as it shows how this architecture plans to achieve self-supervised expansion.

3.1 Model Structure - Rough Prototype Present

Here, the basic components, such as the node and cluster, are described.

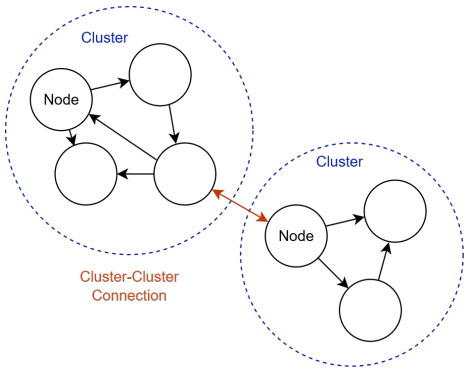

Figure 3: Node and Cluster Layout

A node is just like in conventional ANNs. It has inputs and outputs, which are influenced by the connection nodes. The main difference is in how these are processed. In this architecture, nodes also have a state, which is passed on at each iteration. This allows the network activations to evolve, much like a wave. This wave-like propagation helps maintain temporal continuity, allowing information to be refined over successive iterations.

As for clusters, these are composed of many nodes with defined input and output ports. This allows them to be combined with other clusters or used to feed data into the model. Another advantage is that clusters can be split into parts for GPU computation. This design naturally supports GPU parallelism, since clusters can operate semi-independently on different GPUs with minimal cross-communication.

In this repo, you can find the current basic implementation of nodes and clusters.

GitHub: Leonhard17/SelfExpandingNN

3.2 Other Parts

There is no clear path for code or formulas yet, but the basic outline is in place as described above. For exact details and descriptions, feel free to contact me. I can better explain where this is going in a direct conversation.

I am also always happy to share more technical details or discuss ideas and concerns with anyone interested.

3.3 Self-Regulation and Alignment

For alignment and safety, this system utilizes a mechanism similar to the reward center in our brain. To understand the working principle, we first have to know how our brain evaluates situations and gives reward signals.

I will give an example of how we evaluate situations. Think of a moment that left a deep impression on you, good or bad. We often assume our internal systems value memories based on their true impact or long-term importance. This is how we believe we learn and remember. However, the most important positive or negative moments might not even come to mind, such as the impact on your reputation of always being punctual or completing projects on time.

What we remember best are events that deviate from what we expected, such as trying something new and immediately loving it, receiving an unexpected text, or someone cutting you off on the road. These are not the most important things that happened to you, but they are the most memorable. This sensitivity to surprise is what makes learning an efficient process. It directs attention and adaptation exactly where our expectations fail. In this sense, the network's internal surprise, measured through entropy, plays a role similar to that of our brain's prediction errors in driving attention and memory. Of course, this also comes with risks. A system guided too much by unexpected events might overemphasize coincidences, much like how humans develop superstitions.

I try to do the same for the network. This may not align the model from the outset, but it serves as a consistency mechanism. It encourages the network to reinforce its learned patterns and values, helping it stay anchored even as it adapts to surprises. This is how the network gets feedback from its output. It examines what happens internally and feeds the information back in, enabling it to evaluate its actions and self-train independently. Over time, this should reinforce patterns that line up with its foundational values.

But where is the alignment? That is a tricky question. It depends a lot on the initial model training. This is where its values form, which it then tries to stay attached to. We need to start it off in the right direction and show it

the proper behavior so it becomes self-enforced. I am still grappling with how to keep meaningful oversight over such a self-organizing system, especially as it evolves new structures that could be opaque to us.

Perhaps accurate alignment for such systems will need principles even deeper than the reward-prediction mechanisms we see in biology. This remains a largely open research question, and I would be delighted to hear thoughts from anyone exploring similar challenges using other models. Feel free to reach out.

4 Conclusion

I have developed this system over the last two years and have had only partial success with its implementation, mainly due to a lack of resources in both time and money. But I thought the architecture was worth sharing so it might gain some attention, and the work on this system might continue in some form.

I genuinely believe this architecture has considerable potential to address some of the most significant limitations in AI today. It could address rigid models, the lack of self-evolving systems, and major challenges in computational efficiency. I believe this might be the kind of architectural leap we see only rarely in AI, a chance to explore an entirely new class of systems, not just keep scaling the old ones. And if this model is possible and works, I am not sure it should not be considered conscious. It is very similar to our brain structure and possesses an active internal mind. The question we also have to ask is what makes us different from such systems. Naturally, there is a different kind of information processing, but technically speaking, are we not also just models that pass around information?

I hope you liked this proposed architecture, as it offers a genuine divergence from the current path of AI evolution, which primarily focuses on getting bigger. These ideas instead point toward a completely different paradigm of thinking systems.

Thanks for reading!

5 Call for Collaboration

I would be very grateful for any thoughts you have about this project or for pointing out blind spots I might have missed. Whether you are a researcher, engineer, hobbyist, or just deeply curious about these ideas, I would love to hear your perspective. If you would like to collaborate, discuss ideas, or provide support for the project, please don't hesitate to reach out.

As a young independent researcher, I lack the institutional support and computing resources typically found in major labs. But I have the time, willingness, and ideas to push this forward if given the chance. Funding has been the most significant limiting factor in my pursuit of this research. Even modest funding could allow me to run experiments that are not possible on personal hardware and could rapidly accelerate the testing of this system.

Even small contributions, whether in the form of thoughts, collaborations, or resources, can help bring this radically different kind of AI closer to reality. Ultimately, exploring architectures like this may not only change how machines learn but also how humans interact with them. It could change how we understand learning and consciousness itself.

E-mail: leonhardwaibl@gmail.com

GitHub: Leonhard17/SelfExpandingNN

6 Related References

These are some references I discovered while researching for this project. I used AI to generate concise summaries, allowing you to better understand what each work is about. This isn't a comprehensive bibliography and is provided only to serve as a starting point for interested readers to explore other works.

6.1 Biological Inspiration

Sporns, Networks of the Brain (2011)

Shows how intelligence comes from complex, adaptive, and highly dynamic network topologies, not just weight tuning. This is foundational for why EvoNet is a living graph that restructures itself.

Goldman-Rakic, Cellular basis of working memory (1995)

Explains how persistent activity in prefrontal circuits underlies working memory. Directly supports EvoNet's idea of a persistent internal state.

Turrigiano, Homeostatic plasticity in neuronal networks (1999)

Describes how real neurons balance their activity to avoid both runaway excitation and silence, exactly like EvoNet's entropy-energy control.

Friston, The free-energy principle: a unified brain theory? (2010)

Argues that brains minimize surprise (free energy), which resonates with EvoNet's entropy-based self-supervision.

Rosenbaum et al., The spatial structure of correlated neuronal variability (2019)

Shows how specific recurrent connectivity patterns determine computational capabilities, matching EvoNet's belief that evolving graph structure is key.

6.2 Information Theory and Entropy

Shannon, A Mathematical Theory of Communication (1948)

Introduced entropy as a formal measure of uncertainty, the core of EvoNet's idea of using entropy to guide both learning and structural adaptation.

Hinton van Camp, Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights (1993)

Early work on controlling model complexity through entropy-like objectives, similar to EvoNet minimizing uncertainty to organize itself.

6.3 Uncertainty and Learning Signals

Kendall Gal, What Uncertainties Do We Need in Bayesian Deep Learning? (2017)

Separates aleatoric from epistemic uncertainty. EvoNet mirrors this split, using aleatoric for training and epistemic for expansion.

Schmidhuber, A possibility for implementing curiosity and boredom in model-building neural controllers (1991)

Proposes curiosity-driven learning based on prediction errors, very close to EvoNet's idea of entropy spikes guiding attention and growth.

Lillicrap et al., Random synaptic feedback weights support error backpropagation for deep learning (2016)

Shows that learning can still work with noisy or asymmetric feedback, background for EvoNet's local entropy-driven adjustments.

6.4 Graph-Based and Sparse Architectures

Battaglia et al., Relational inductive biases, deep learning, and graph networks (2018)

Argues that graphs naturally capture complex relationships, just like EvoNet's non-layered, evolving graph design.

Shazeer et al., Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer (2017)

Selective activation of different experts based on input, similar to EvoNet's clusters selectively engaging and growing.

6.5 Dynamic Architectures and Neuroevolution

Stanley Miikkulainen, Evolving Neural Networks through Augmenting Topologies (NEAT) (2002)

Classic work on evolving both weights and network structure, conceptually close to EvoNet's continual self-expansion, though done across generations.

Stanley et al., Designing Neural Networks through Neuroevolution (2019)

A broader overview of evolving neural architectures, emphasizing the power of letting structure adapt.

Mitchell, Self-Expanding Neural Networks (SENN) (2024)

One of the most directly related modern ideas — SENNs grow their structure as they learn, closely paralleling EvoNet.

6.6 Architecture Search

Zoph Le, Neural Architecture Search with Reinforcement Learning (2017)

Used RL to find optimal structures before deployment. Not dynamic after training, but shows how powerful changing topology can be.

Liu et al., DARTS: Differentiable Architecture Search (2019)

Gradient-based NAS that speeds up finding better architectures. Relevant as it shows structural adaptation is usually better than static scaling.

Elsken et al., Neural Architecture Search: A Survey (2019)

Broad survey of NAS techniques, making the case that searching for better architectures often beats just adding parameters.

Note: I wrote this entirely myself, only using AI tools for grammar correction. All core ideas, structure, and text are my own.