Communicating Research to the General Public

The WISL Award for Communicating PhD Research to the Public launched in 2010, and since then over 100 Ph.D. degree recipients have successfully included a chapter in their Ph.D. thesis communicating their research to non-specialists. The goal is to explain the candidate's scholarly research and its significance—as well as their excitement for and journey through their area of study—to a wider audience that includes family members, friends, civic groups, newspaper reporters, program officers at appropriate funding agencies, state legislators, and members of the U.S. Congress.

WISL encourages the inclusion of such chapters in all Ph.D. theses everywhere, through the cooperation of PhD candidates, their mentors, and departments. WISL offers awards of $250 for UW-Madison Ph.D. candidates in science and engineering. Candidates from other institutions may participate, but are not eligible for the cash award. WISL strongly encourages other institutions to launch similar programs.

Wisconsin Initiative for Science Literacy

The dual mission of the Wisconsin Initiative for Science Literacy is to promote literacy in science, mathematics and technology among the general public and to attract future generations to careers in research, teaching and public service.

Contact: Prof. Bassam Z. Shakhashiri

UW-Madison Department of Chemistry

bassam@chem.wisc.edu

www.scifun.org

DISTRIBUTING DEEP NEURAL NETWORK INFERENCE ACROSS EDGE-HUB-CLOUD SYSTEMS

by

Robert Viramontes

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2025

Date of preliminary examination: May 7, 2025
The dissertation is approved by the following members of the Final Oral Committee:
Azadeh Davoodi, Professor, Electrical and Computer Engineering
Umit Ogras, Professor, Electrical and Computer Engineering
Jeffrey Linderoth, Professor, Industrial and Systems Engineering
Mahesh Sharma, Fellow, Tenstorrent

10 CHAPTER FOR THE PUBLIC

In this chapter, I present my research in a form geared towards a general public audience. My work has been supported by the public through programs like the National Science Foundation and I hope to make the results of my work more accessible. Access to knowledge is a powerful tool in developing an engaged and compassionate society, and I am proud to contribute in my own small way. Many thanks to editor Elizabeth Reynolds for thoughtful remarks, and to Professor Bassam Shakhashiri and Cacye Osborne at the Wisconsin Initiative for Science Literacy for supporting this program!

10.1 Background on Artificial Intelligence (AI)

Nowadays, many of us are familiar with interacting with artificial intelligence (AI) programs in our everyday life. You might use Grammarly to help draft and revise an email or Google's Gemini to generate a logo for your sports team. All of these modern AI tools rely on a fundamental invention: the neural network.

The neural network is, at least in computing terms, an ancient idea with the first generation developed in the 1950s. As you might suspect based on the word "neural", these computer programs are inspired by the brain and use digital "neurons" to process information. Much like our brains process what our eyes see, early neural networks focused on processing pictures to recognize handwriting.

Many factors contribute to the success of modern AI tools, but one of the key insights, which occured around 2012, was "deep" neural networks (DNNs). I like

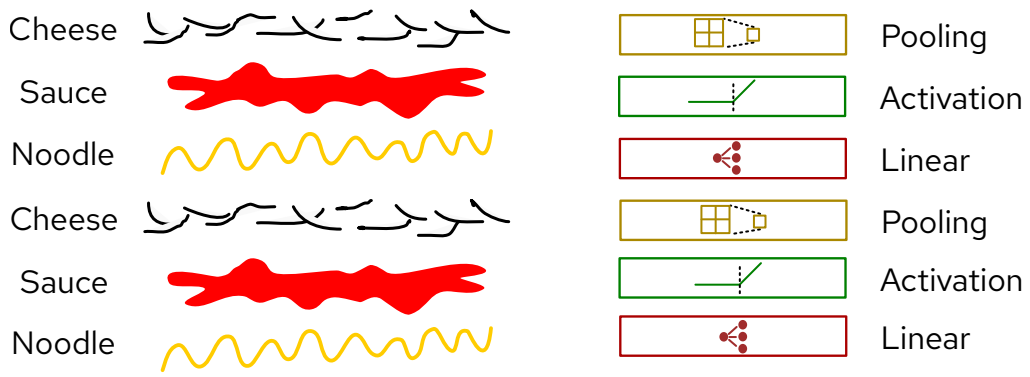

Figure 10.1: A lasagna (left) is a bit like a deep neural network (right). Both are composed of multiple layers of different types, with each contributing something important to the final outcome.

to think about this like one of my favorite foods, lasagna! Similar to a lasagna which might have layers of noodle, sauce and cheese, a DNN uses different layers of operations for different effects. Also like a lasagna, a DNN of one layer isn't particularly good, so we build up multiple layers to improve them!

As the lasagna gets more and more layers, we need larger and sturdier pans to bake in. If we have hundreds of layers, we might even need to look at moving to some industrial oven equipment. The story is similar for DNNs, where small models with few layers might work on our phones or laptops, but the latest and greatest (and largest) require industrial-sized computers in datacenters.

As we have watched the rapid evolution of AI tools in the past few years, I have been very concerned about the impact of their high energy use and potential for carbon emissions. Understanding how to minimize these impacts has been a very interesting question to explore, both from a technical perspective and its social merits. My approach to this problem has been to split those layers up to utilize multiple computers for DNNs. It would be like baking layers of the lasagna

separately, and only stacking them up at the end when it's ready to serve. I call this **layer assignment**, trying to find which layer is best assigned to which device. My research has shown that we can use this technique to improve latency (speed), energy (battery life), privacy (keeping control of your data), and carbon emissions (sustainability).

10.2 What's in a Layer

Before we can begin assigning layers to a computer, we need to know a bit more about what's happening in each layer. I primarily consider the latency to compute the layer and the energy to compute the layer, key metrics in many computer systems.

- **Latency** is the amount of time it takes to compute a layer from start to end. This is measured in milliseconds.
- **Energy** is the amount of work that the computer must do to compute a layer from start to end. This is measured in joules, though you may also be familiar with measures like kWh on your utility bill or mAh on mobile phone batteries.

There are two typical ways to collect this data: using formulas to estimate and directly measuring latency and energy on the device. For either, it is import to have a rigorous and repeatable method because we need to measure for each computing device in our system and the values must mean the same thing from device to device. For instance, a smartphone will usually be slower than a desktop and might

use less energy, but we want to make sure that isn't because of an error in the test methods. Overall, I prefer to use the direct measurement approach because it tends to be more accurate, but the formula approach can be useful for prototypes.

Figure 10.2 shows the setup in my lab for collecting latency and energy information for an NVIDIA Jetson Nano. This setup includes the actual device, the NVIDIA Jetson Nano, a SmartPower 3 power supply that I use to gather the energy data, and a PC for logging and analyzing the data. You can see why setting this up for every device is more complicated than plugging a few numbers into a formula!

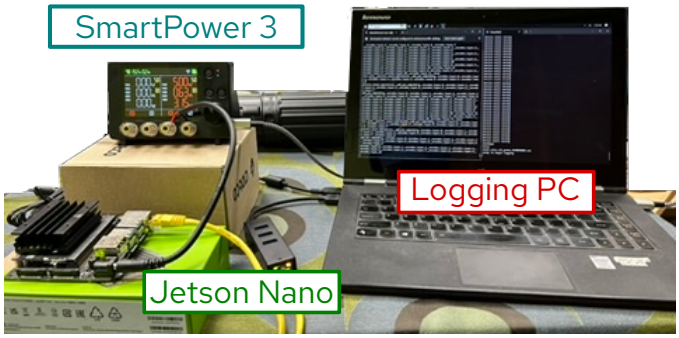

Figure 10.2: The lab setup for measuring latency and energy used by the NVIDIA Jetson Nano.

10.3 What's the Deal With All These Computers?

So what about the computers that I'm assigning these layers to? Imagine a theoretical lasagna factory, like in Figure 10.3. This factory might have "smart" cameras that monitor different conveyor belts, some local computers for running the factory, and Internet access to cloud computers.

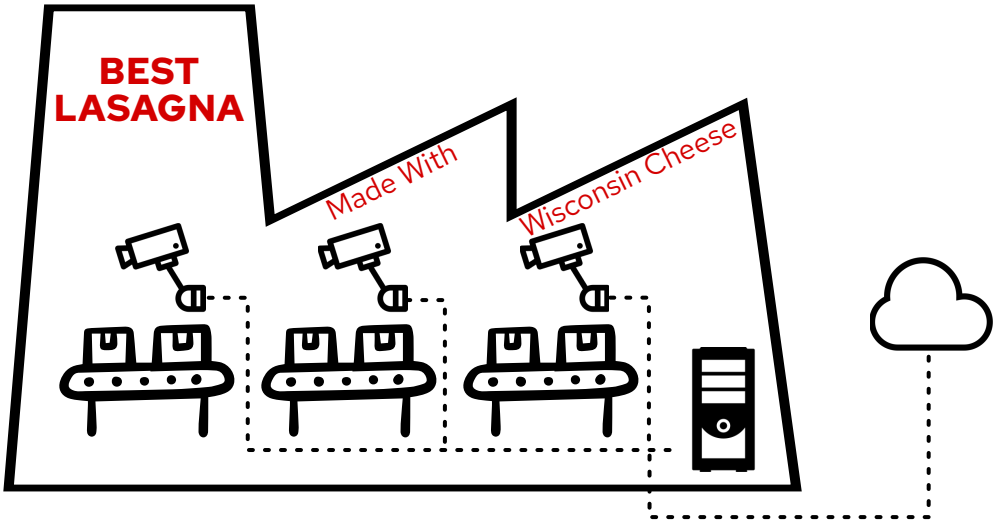

Figure 10.3: Theoretical lasagna factory demonstrating the various types of computing devices in a system and how they can communicate.

In my research, we call the smart cameras "edge" devices, the factory computers "hub" devices, and the external cloud "cloud" devices. These are "small", "medium", and "large" computers, respectively. Each of these devices has the ability to do at least some of the work of computing the DNN. They are also all connected in such a way that they can all talk to each other.

10.4 Useful Objectives

Now, we have to decide what to do with all of this information we've collected about layers. I want to improve something about the AI system, so I want to express an *optimization* goal. This is usually expressed as *"minimize A subject to B"*, which means that we want to reduce A while staying within limits of B. "Subject to" is optional, but often very useful as we'll see. Say we wanted to bake a lasagna faster.

One approach could be to use only one thin layer, but as I said before, that's not a very good lasagna. So we could say *"minimize bake time subject to having four layers"*, so we still have a good lasagna. We could then change oven temperature, baking dish size, and baking dish material to minimize the bake time.

For DNNs, over the course of my research, I've identified a few useful optimization goals when assigning layers:

1. Minimize latency: this minimizes the amount of time it takes to get an answer back. This improves the user experience by giving a more interactive experience with less wait time for the computer to "think".

2. Minimize latency subject to privacy: a tweak of the previous goal, this minimizes the amount of time to get an answer while providing additional data privacy guarantees. This further improves the user experience by reducing the amount of identifiable information shared.

3. Minimize energy subject to latency: this minimizes the amount of work put in to compute the answer. The latency constraint is important to guarantee that the user gets an answer back in a timely manner (as a professor I know likes to say "minimizing energy is easy: just turn it off!"). This improves the user experience through reducing energy bills and improving battery life.

4. Minimize carbon subject to latency: a tweak on the energy goal, we look at carbon emissions as a sustainability metric. This improves user experience by reducing the environmental impact of running AI.

10.5 Finally, Some Results!

Preheating with Preloading

Over the course of my PhD, I've explored various approaches to achieve these objectives. In my first project, we only looked at reducing latency. While investigating the various causes of latency, we found a significant source we could hide with a clever layer assignment. The layers in the DNN usually have some extra data associated with them, called "weights", that can take a long time to load. We proposed that we could hide this by starting this loading process early, while another device was busy computing another layer. We call this **weight preloading**.

In the DNN, the layers use the result of the previous layer. This means they must be executed in a particular order, and only after the previous layer is finished. This is a bit like if you couldn't start on the next layers of lasagna in a second pan until the first layers are done baking. However, with our layer assignment strategy, we know before we start baking which layers will go in which pan and what order the pans go in the oven. This means we can start building the layers while the previous pan is baking, so it is ready to go into the oven right when the previous pan comes out. While we still can't fit multiple pans in the oven, we've overlapped the baking time of the previous layer with the preparation time of the next layer to hide that preparation time.

We explored this weight preloading idea with different DNNs and different computing systems. In these experiments, we found a few interesting results:

- When the devices are good at talking to each other (the network has high bandwidth), it's easy for the smart cameras to send all the work to the cloud. Weight preloading isn't very useful in this case, because there aren't opportunities to overlap compute and weight loading (the industrial oven can bake all the layers at once).

- When devices are bad at talking to the cloud (the network has low bandwidth), the small and medium computers tend to talk to each other quite a bit. Their limited resources makes weight preloading more appealing (smaller ovens means more opportunities to overlap baking and preparing).

- We also found that certain types of layers within the DNNs had more benefit than others. Layers that had many weights to load were the best for hiding preparation time with weight preloading.

If systems implement our weight preloading technique, the latency of DNN computation can be reduced significantly. This will lead to AI applications that respond to users more quickly, particularly in areas with low Internet access.

Baking with Bundles

Over the course of my research, I have learned a lot about measuring latency and energy on these devices. This became the focus of my second project. In this project, we examined some problems with what we referred to as **bundle-based profiling**. When measuring the latency and energy for the layers of our DNNs, we want to collect data about the individual layer as well as groups of layers. This is because

determining the latency and energy of a group of layers is not a simple sum of its parts. This is a bit like if you were to try to bake four layers of lasagna in one pan with four layers or four pans with one layer in each. In the second case, every time we change pans, we might experience things like the oven cooling a bit when the door opens and we need to heat the pan as well as the layers. These *external factors* cause baking four layers in one pan to be very different than one layer each in four pans.

However, we found that the situation was a little bit more complicated for DNNs. On some devices, grouping the layers into a bundle would reduce the latency; on other devices, bundles increased the latency. This becomes a problem when we want to optimize, for instance, latency. During this optimization process, we look at all posssible combinations of layers, and pick the combination with the smallest latency. For a device whose bundle is *larger* than the sum of its parts, it looks better to pick individual layers instead of the bundle. But this is a side effect of the mathematical expression, and does not represent reality.

This would be a bit like if, for baking the four layers, we were told to bake all four layers in the same pan. You and I, actually in the kitchen, would build all four layers and put this into the oven at once, forming a bundle of 4 layers. But the recipe writer, wanting to give the lowest bake time estimate, instead used the sum of baking the four layers in the same pan separately. We would be very disappointed with the cold center of our lasagna!

Unfortunately, this problem took quite a bit of profiling effort to uncover and understand. Fortunately, the solution was quite straightforward. We could add

a check during our optimization search that ensures, for layers assigned to a device, the bundle values are used instead of the individual layer values. When we implemented this, we found:

- Without bundles enforced, the latency can be mis-predicted. This causes the layer assignment to be incorrect.
- In one case [1], the actual result is 1.2x slower than the estimate. This would be like a recipe that guaranteed a cold center.
- In another case [2], the actual result was 1.3x faster than the estimate. This would be like a recipe that guaranteed burnt lasagna.
- By considering bundles, we guarantee that the layer assignment is correct in mirroring real-world conditions.

Improving DNN latency estimation will allow developers to make better decisions about layer assignment. This will encourage more consistency and fewer surprises when trying to optimize the experience for users.

Resting with Efficiency

One of the big discussions around AI has been about the massive energy requirements for maintaining these computers. This is a significant issue with broad effects, and I'm glad my next project was able to focus on this. To do this, we exploited an

[1] Table 7.1 BW=1E5
[2] Table 7.2, BW=1.375E5

interesting feature on modern devices, *frequency scaling*. Frequency scaling allows us to control the operating frequency of the device, or how fast it gets work done. This is kind of like oven temperature: a high temperature may cook faster but also increases the chances of burnt edges while a low temperature cooks slower and more evenly, but too slow and you'll need a snack before dinner is ready. There is a "sweet spot" for oven temperature.

Similarly, for a computer operating frequency, a higher frequency means it can get work done faster but consumes a lot of energy. A lower frequency means the work gets done slower, and may be done so slowly that the energy starts going up. For energy efficiency, there is often a "sweet spot" setting for the device operating frequency.

Up until now, I had profiled the devices for their latency and energy utilizing the default settings (imagine just hitting a "lasagna" button on the oven and hoping it picks the right temperature). Now, I added manual control of the operating frequency to find energy-efficient solutions. The goal was "*minimize energy subject to a maximum latency*". It is important to add that we require a minimum latency, or else we might pick a very slow operating frequency and the user gets frustrated when the device is slow to respond. We also explore "*minimize latency subject to a maximum energy*", which would be useful for mobile phones where we want fast responses but don't want to use too much battery life. Finally, we explored privacy preservation, with the goal "*minimize latency subject to keeping some layers private*", which reduces the risk of sending personal information to a third party.

With this new knob of control, we ran experiments and came to some interesting conclusions.

- Compared to picking just the fastest or just the slowest setting, our method finds the "sweet spot" operating frequency somewhere in the middle to optimize our goal.
- The "sweet spot" changes depending on which DNN we use. This would be a bit like different styles of lasagna, with different number of layers, having different oven temperature settings.
- The "sweet spot" also changes depending on the goal, whether it be reducing energy, reducing latency, or preserving privacy.
- Devices have built-in features that try to automate this, but they have a more guess-and-check approach. Our technique that has complete information about the DNN consistently improves or matches the result.

This system allows for us to decide "how" to run a DNN layer in addition to "where". By adding this, we can improve control over latency and energy use of AI.

Serving with Sustainability

In my last major project, instead of focusing on energy, I focused on carbon emissions. Carbon emissions is a key metric in assessing sustainability because it doesn't just measure the quantity of energy, but the quality of that energy.

Carbon emissions can be calculated from the energy using a number called the carbon intensity, which gives the carbon emissions per unit of energy. The key thing is that carbon intensity varies over time, both throughout the day and over the course of the year. A common example is an electricity grid that has solar panels, which provide low-carbon energy, as one source. During the day when the sun is out, the grid's carbon intensity is lower and it rises in the evening as the sun sets. Similarly, the carbon intensity is lower in the summer when the sun is more intense and rises in the winter.

Carbon intensity is also tied to a specific electric grid that services a particular geographic location. In our experiments, we try to find the carbon-optimal layer assignment when devices are in different grids. We "*minimize carbon subject to a maximum latency*", to find a sustainable solution without annoying the user with slow responses. This might be like if we had access to multiple kitchens, one with a traditional oven and one with convection oven. Even though the temperature is the same, the quality of the heat is different. In what cases does it make sense to swap between ovens? How does that change if the other oven is just downstairs or across town?

During our experiments, we used data from different grids including grids in the Midwest and California. As a result, we found:

- When starting in a high-carbon grid, it makes sense to share the work and utilize the low-carbon grid for at least some layers.
- As carbon intensity varies throughout the day, it's possible to achieve lower latency and lower carbon in some cases. Even though lower latency usually

means higher energy, the layer assignment can take advantage of low-carbon areas to mitigate this.

- Energy-optimal and carbon-optimal solutions can differ in many cases. It is not enough to consider energy alone, but we do have the tools to consider carbon and optimize for sustainability.

By considering various power sources, we can direct layer assignment to reduce the carbon emissions. This helps to mitigate sustainability concerns about the growth of AI applications.

10.6 Conclusion

As AI has been rapidly adopted in many aspects of our daily life, I have been really motivated to explore opportunities to improve the systems that are used to implement these. I've shown that thinking about these systems can lead to tangible benefits, including getting faster answers (reducing latency) and preserving privacy of user data. I'm also concerned with the growing energy demands for running AI, and have shown that layer assignment can be an effective tool in addressing energy and sustainability concerns. I am very optimistic about the future of AI developments and believe there is an exciting opportunity to keep improving the computing systems; I hope the reader joins my excitement!