Blaize® Graph Streaming Processor®

The Revolutionary Graph-native Architecture

The Blaize® Graph Streaming Processor® (GSP) represents a significant inflection point for the industry. Architected for graph-native AI application development, it represents a fundamental way to change how we compute the intense workloads of the future.

Neural Networks Change Computing

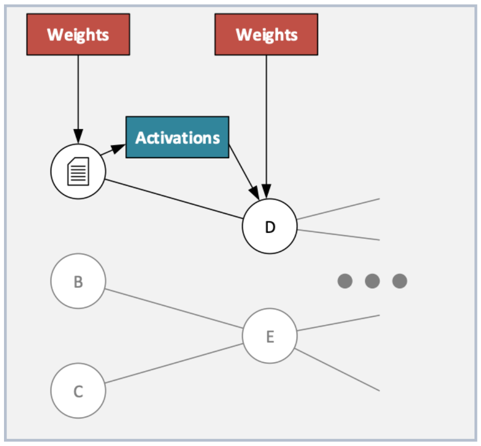
Artificial Intelligence (AI) has taken the world by storm, in many cases surpassing human-level accuracy in areas like image recognition and speech recognition. The convergence of big data, fast hardware processing and advances in AI algorithms have accelerated the adoption of AI in almost every industry. One of the core advancements driving this surge in AI is the emergence and evolution of neural networks, software models that are intended to mimic (in a simple fashion) how neurons in the brain behave.

Neural networks are extremely powerful, but getting these neural networks deployed into real-world applications has many challenges:

- Neural networks need to run efficiently close to where real-world data is collected - environments where computing, memory and energy resources are constrained.
- Neural network architectures, models and methods evolve very rapidly, and developers need to be able to update their models easily and quickly.
- Neural networks are usually part of an overall AI application, so developers need to integrate non-neural network functions along with neural network functions in order to deploy their applications in the real-world.

There are different neural network architectures to accommodate the varied and wide range of AI tasks, but every neural network has one common feature: they are all structured as graphs.

The high-level frameworks used for creating and training models are already graph-oriented, but the underlying hardware used for inference have been stuck in the past. Much of the development effort in implementing a NN model involves adapting the graph to a non-graph architecture. This means both longer development cycles and less efficient processing in the finished product.



Neural networks consists of multiple layers, each of which has multiple nodes. Processing starts at one end of the network, proceeds through all of the layers, and terminates at the other end of the network. Within each node resides a small computer program, sometimes referred to as a kernel. The kernel performs the functions of that node, manipulating of weights and inputs. The outputs of one layer, also known as activations, become the inputs of the next layer. Each layer is designed to extract some feature, for example lines or shapes from images, and as the layers progress the neural network graph gets built to progressively extract higher and higher level features to ultimately identify something in the input, like objects in an image.

Architectural Challenges of Processing Neural Networks

Existing processors such as CPUs, GPUs, FPGAs, DSPs as well as newer fixed function solutions were not built to process neural network graphs natively. When processing neural networks, different architectures encounter different challenges:

Hitting the memory wall — traditional architectures which are not graph-oriented cannot leverage the structure of the graph and the relationships between nodes. So, when processing each node in the graph, they must send intermediate results to external memory and then later fetch it back for processing the next node. This creates a significant amount of extraneous data movement, limiting performance, increasing latency and increasing power consumption.

Lack of true task parallelism — Neural network processing lends itself well to four different levels of parallelism: instruction-level, data-level, thread-level, and task-level. The first three are available in some form in most current architectures, but none of them offers task-level parallelism. True task parallelism allows multiple nodes and multiple threads to be in play at any given time with balance in the production and consumption of data through the graph, enabling efficient streaming of the graphs. Streaming eliminates a huge amount of data movement, saving power and time that would have been needed to move the data in a different architecture.

Programmability — different architectures support different levels of programmability, from high level software programmability to hardware-level programmability. High level abstraction in software programmability allows developers to build more complete applications, but limits the efficiency of hardware. Hardware-level programmability requires detailed hardware knowledge and can take a long time to implement, and this approach takes too long to keep up with the innovations in neural network architectures. None of the existing architectures allows developers to program at a high-level graph in software but also have that programmability translate all the way through to the hardware architecture efficiently.

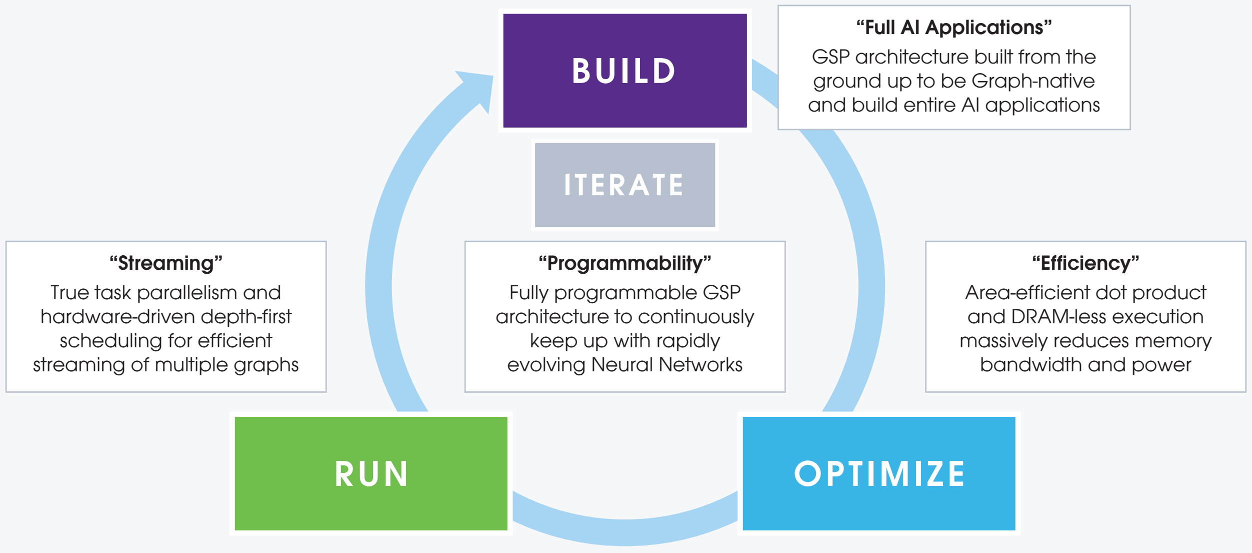
Edge Solutions Must Be Updatable

A unique characteristic of machine learning (ML) is that models are fluid, remaining so even after deployment. The models are very sensitive to the training sets — even the order of the members of that training set. As development of an ML-enabled system proceeds, and even after the system is deployed in the field, the model is likely to be improved many times. This calls for system flexibility so that the model can be updated easily both during development and after release. Such flexibility requires full programmability.

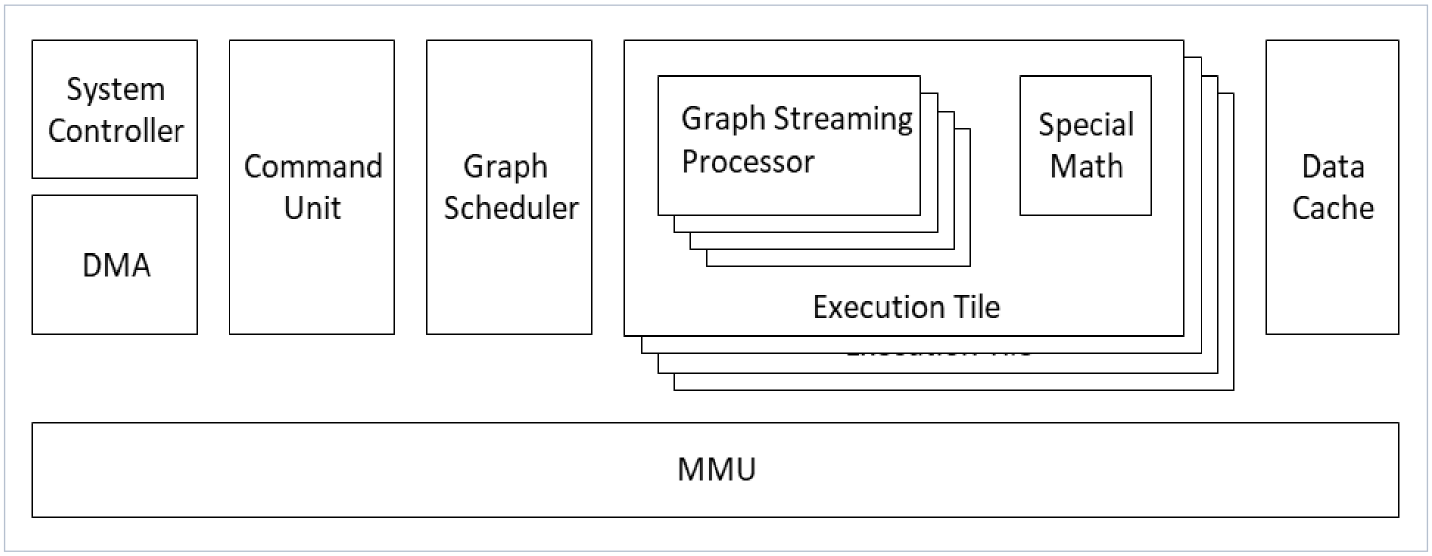
Graph Streaming Processor: Efficiency & Programmability

The Graph Streaming Processor architecture, or GSP, is the first true Graph-native architecture built to address the challenges in efficiently processing neural networks and building complete AI applications. With a fully programmable graph streaming architecture, GSP chips process graphs more efficiently than CPU/GPU architectures. As a result, the GSP architecture enables developers to build entire AI applications, optimize these for edge deployment constraints, run these efficiently in a complete streaming fashion, and continuously iterate to keep up with rapid evolutions in neural networks.

Graph Streaming Processor Architecture Enables Graph-Native AI Application Development


BUILD
ITERATE
RUN
OPTIMIZE
"Full AI Applications"
GSP architecture built from the ground up to be Graph-native and build entire AI applications
"Streaming"
True task parallelism and hardware-driven depth-first scheduling for efficient streaming of multiple graphs
"Programmability"
Fully programmable GSP architecture to continuously keep up with rapidly evolving Neural Networks
"Efficiency"
Area-efficient dot product and DRAM-less execution massively reduces memory bandwidth and power

High level block diagram of the GSP architecture



The GSP architecture consists of an array of graph streaming processors, dedicated math processors, hardware control and various types of data cache. The 4 main architectural characteristics of the GSP that enable efficient graph streaming of entire AI applications are:

- True task-level parallelism
- Minimal use of off-chip memory
- Depth-first hardware graph scheduling
- Fully programmable architecture

Task-level Parallelism: Highly Efficient Streaming of Neural Network Graphs

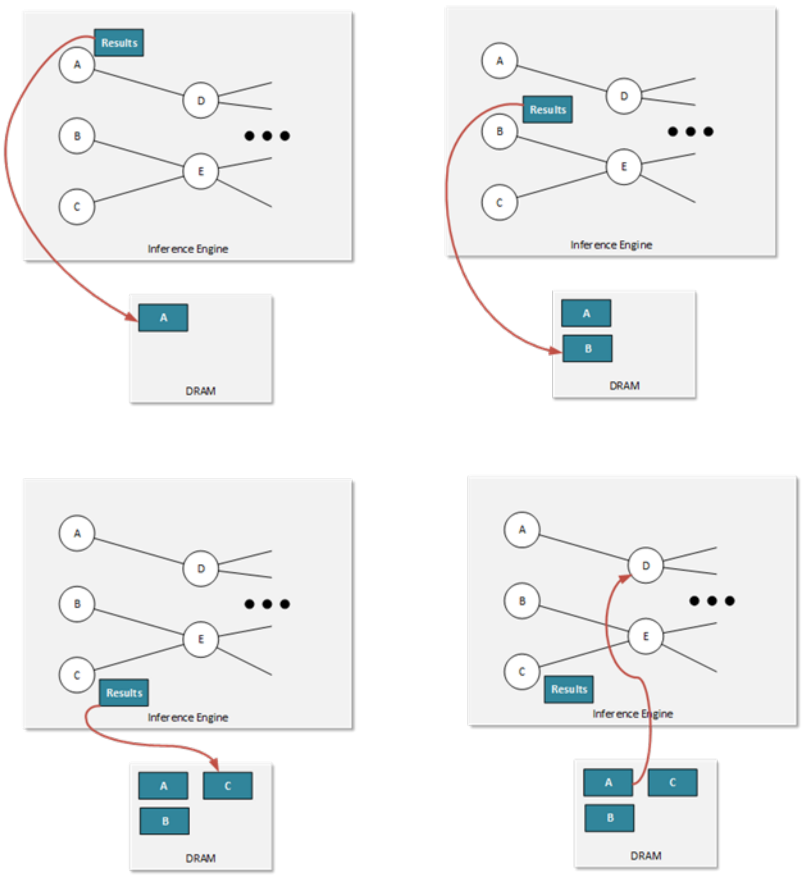
Neural network processing lends itself well to four different levels of parallelism: instruction-level, data-level, thread-level, and task-level. The GSP architecture is the only one in the industry to provide all four levels of parallelism:

- GSPs offer instruction-level parallelism by scheduling instructions as soon as their dependencies have been met. Each processor executes its instructions independently of any other processor.
- GSPs offer thread parallelism, with multiple hardware threads per processor. Changing context either by switching threads or by dispatching a new thread can be done in a single clock cycle.
- GSPs offer data parallelism through special instructions that can operate directly on unaligned blocks of data in a register file. These instructions include 2D block instructions, like block move and block add, as well as reduction instructions, like dot product.
- The GSP, uniquely, provides true task-level parallelism. Multiple nodes from multiple layers can be processed concurrently, and nodes can be scheduled as soon as the data they need has been calculated. They need not wait for the completed processing of any other nodes. Scheduling is dynamic, allowing it to adapt to actual results as they unfold, enabling truly efficient streaming of the neural network graphs.

The GSP architecture supports true task-parallelism enabling highly efficient streaming of neural network graphs. This eliminates a huge amount of data movement, saving power and increasing performance dramatically.

Less DRAM execution: Reduced memory bandwidth and power consumption

The traditional execution of neural network graphs does not leverage the information that the graph can provide. Nodes are calculated fully, one node at a time, without taking into account the relationships between nodes. Because of this, all of the activations from one node — amounting to millions or more of values — must be stored for use as inputs to another node that will be calculated in the future. Because the amount of data to be stored is so enormous, it must be sent off-chip to external DRAM. When needed for another node, the data must then be recalled and brought back onto the processing chip. This data movement burns an incredible amount of power, as well as adding latency that reduces performance. This need to move volumes of data has become the limiting factor in processing neural networks, sometimes being referred to as "hitting the memory wall." (see image below)



The graph-native nature of the GSP architecture dramatically minimizes this data movement back and forth to external DRAM. Only the first input and final output are needed externally, while everything else in the middle is just temporary, intermediate data. The result is massively reduces memory bandwidth and power consumption. An example of this execution for a Convolutional Neural Network is shown below.

Depth-first hardware graph scheduling: Dynamic execution

The GSP architecture includes a hardware graph scheduler that can look ahead in the graph and schedule computations as soon as enough data is available. This allows the GSP to schedule the execution of the graph dynamically, enabling true data flow streaming of graphs, including sparse and conditional graphs. When processing the graphs, instead of processing an entire image (or layer) at a time, the Graph Streaming Processor divides the image into blocks and schedules the computations on these blocks depth-first using the hardware graph scheduler. A simple example below illustrates this depth-first hardware graph scheduling. In figure 1, once 4 add operations have been completed, in figure 2 the hardware graph scheduler dynamically schedules a DOTP operation.

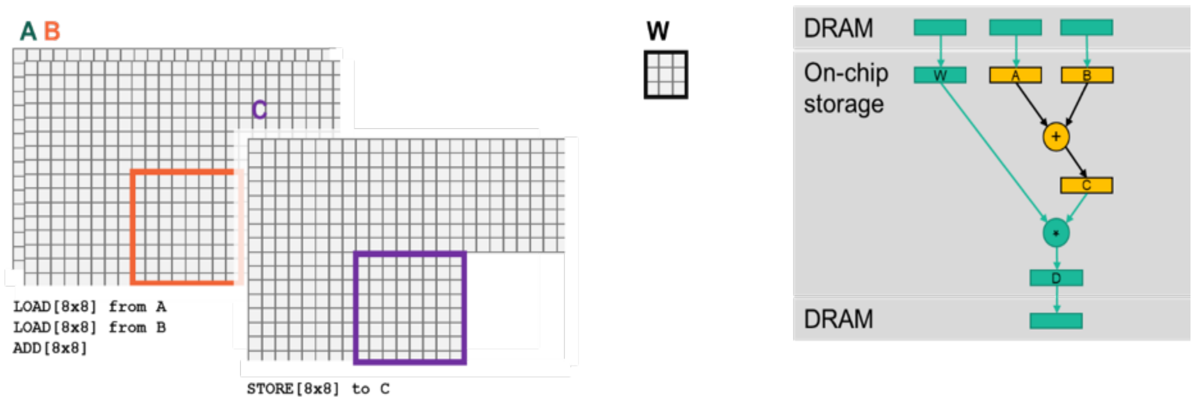


Figure 1: 4 ADD operations completed

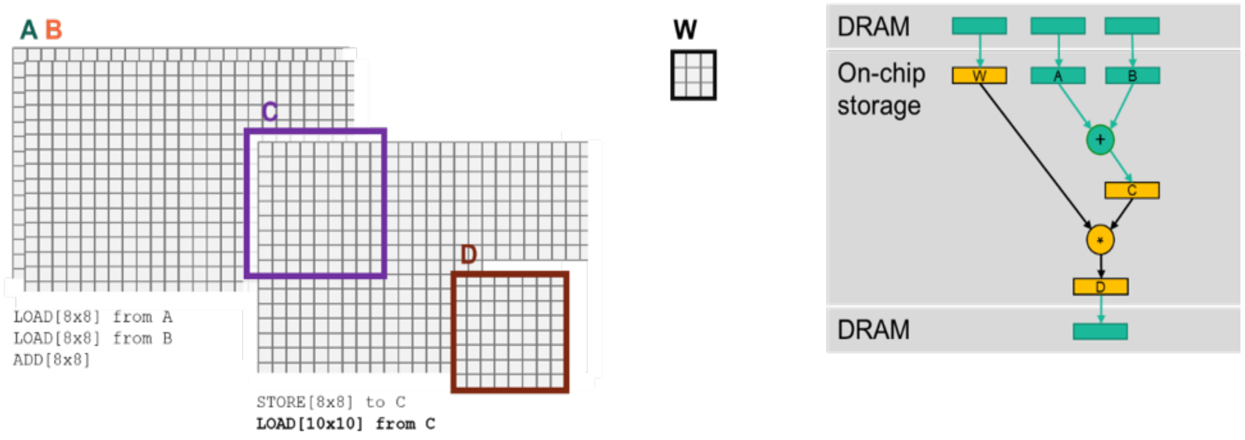


Figure 2: Hardware graph scheduler dynamically schedules a DOTP operation

The result is nodes are scheduled as soon as the data they need is calculated. They need not wait for the completed processing of any other nodes. Scheduling is dynamic, adapting to actual results as they unfold. The hardware scheduler knows precisely when one layer has produced enough blocks for the next layer to be able to consume. Doing so ensures that intermediate data is used as soon as available, and when a block is no longer needed it is discarded from the cache.

Fully programmable architecture: Efficient end-to-end applications

AI developers today, face 3 major challenges when deploying AI applications in the real-world:

1. Neural network functions are usually part of an overall application, and developers need to be able to integrate both the neural network and non-neural network functions easily and efficiently.
2. The entire AI application must run efficiently on the hardware deployed, with high performance, low latency and low power.
3. It is crucial to be able to update applications and neural networks as more data is collected in the field and new neural network innovations rapidly evolve.

The GSP architecture is fully programmable, which enables developers to build end-to-end AI applications in software with deep programmability and efficiency in the hardware as well.

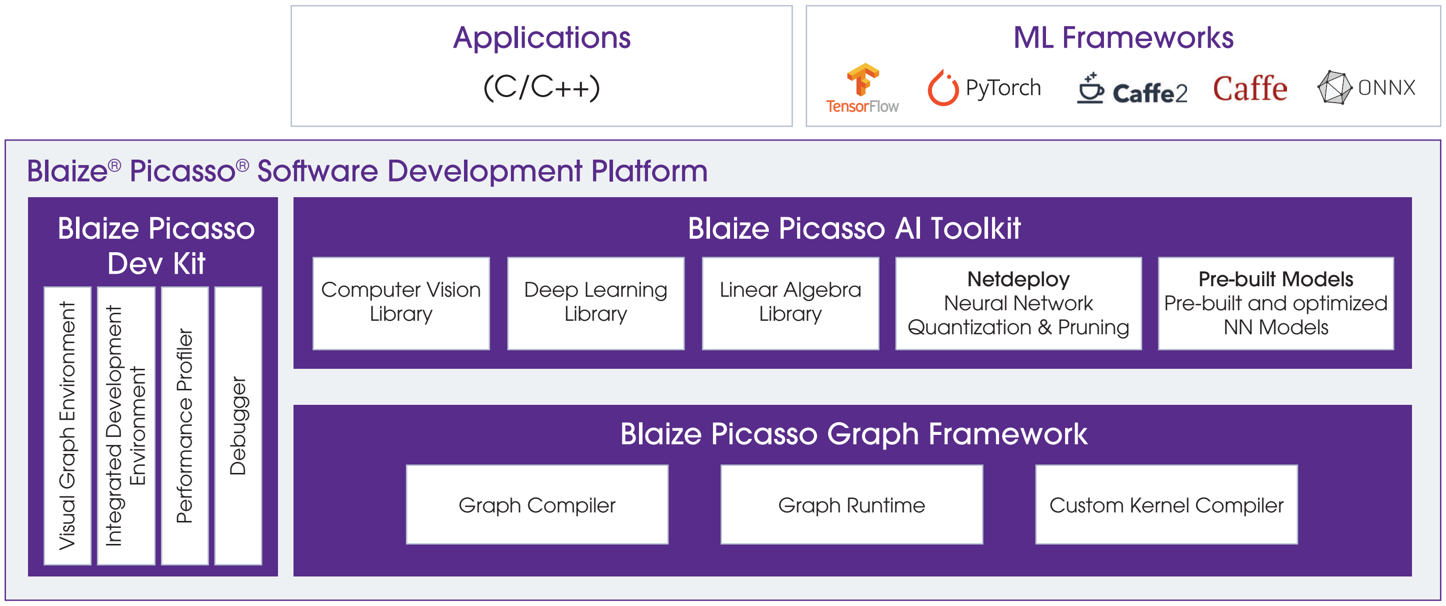
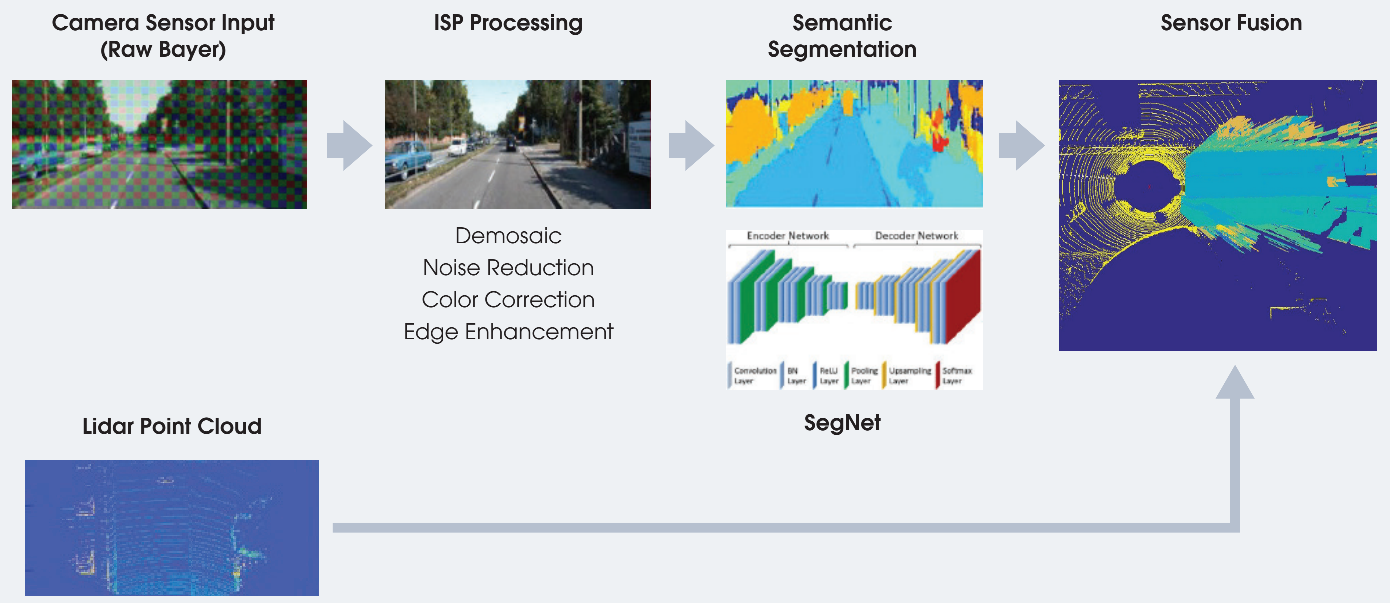


Figure 3: Blaize Picasso Software Development Platform

With the GSP architecture and the Picasso software development platform, developers can build end-end AI applications, integrating non-neural network functions such as Image Signal Processing along with neural network functions that are built in any Machine Learning framework. These graphs are translated efficiently through the Graph framework to run efficiently on the GSP. Thus, the fully programmable nature of the GSP architecture allows AI developers to build entire entire AI applications at the graph-level, run these efficiently on the GSP and easily update the applications as neural networks and workflows evolve.

Complete Automotive Sensor Fusion Application



This application consists of non-neural network Image Signal Processing (ISP) functions, neural network processing for semantic segmentation and functions for sensor fusion, which can all be represented as graphs, integrated together to build the full application and run entirely on the GSP architecture efficiently.

A New Way to Compute Neural Nets

The GSP architecture opens up new possibilities for more efficient, lower-power neural-net processing. Drastically reduced data movement both lowers power and improves performance. Task-level parallelism allows multiple nodes from multiple layers to be processed concurrently. Developers can work at the graph level, making development more intuitive and helping to speed AI solutions to market more quickly with the help of a broader range of engineers. And, because it is fully software programmable, changes to trained models can be easily incorporated during development and in the field.

blaize®

4659 Golden Foothill Parkway, Suite 206, El Dorado Hills, CA 95762 USA
www.blaize.com | info@blaize.com