Performance Implications of Multi-Chiplet Neural Processing Units on Autonomous Driving Perception

Mohanad Odema, Luke Chen, Hyoukjun Kwon, Mohammad Abdullah Al Faruque
University of California, Irvine, USA
{modema, panwangc, hyoukjun.kwon, alfaruqu}@uci.edu

***Abstract*—We study the application of emerging chiplet-based Neural Processing Units to accelerate vehicular AI perception workloads in constrained automotive settings. The motivation stems from how chiplets technology is becoming integral to emerging vehicular architectures, providing a cost-effective trade-off between performance, modularity, and customization; and from perception models being the most computationally demanding workloads in a autonomous driving system. Using the Tesla Autopilot perception pipeline as a case study, we first breakdown its constituent models and profile their performance on different chiplet accelerators. From the insights, we propose a novel scheduling strategy to efficiently deploy perception workloads on multi-chip AI accelerators. Our experiments using a standard DNN performance simulator, MAESTRO, show our approach realizes 82% and 2.8× increase in throughput and processing engines utilization compared to monolithic accelerator designs.**

I. Introduction

The landscape of the automotive industry is being transformed through software-defined vehicles (SDVs) that enable integrating sought-out features of Advanced Driver Assistance Systems (ADAS), autonomy and infotainment (AR/gaming) [1]–[4]. To maintain flexibility and manage rising compute demands, emerging vehicular systems are shifting towards new cross-domain, centralized Electrical/Electronic (E/E) architectures with a few powerful processing computers and zone ECUs [1], enabling easier integration of new features and updates, and allowing automakers to adopt customized chip design tailored to their needs. As a result, automakers like Tesla, GM Cruise, Volkswagen have entertained the adoption of customized System-on-Chips (SoCs) for their automotive compute requirements [5]–[7]. Still, as automotive AI workloads continue to evolve and rise in complexity, advancements in automotive SoC hardware also become a necessity, and that presents a considerable challenge given the long design cycle, high manufacturing costs, and Moore's law stagnation.

Chiplets technology presents a viable solution for the automotive industry [2], [4], [8], [9]. Owing to chiplets' composability property, a scalable, customizable approach becomes feasible supporting the integration of individual hardware modules on the package level, potentially from different technology node generations, and enabling individual component updates to be incorporated with a faster turnaround time than monolithic SoC approaches. Even more so, recent advancements has seen Neural Processing Engines (NPUs) – integral to autonomous driving systems – implemented through consolidating multiple small accelerator chiplets on the package to form a scalable AI inference engine [10]–[15], providing flexible means to control different design parameters (accelerator chips number, connection topology, and heterogeneity) [13], [14].

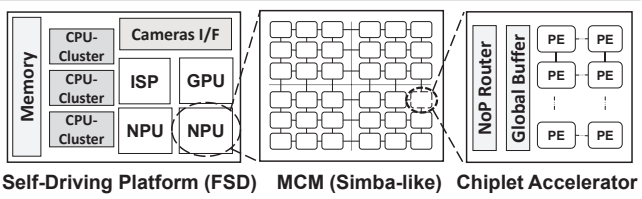

Fig. 1. A descriptive schematic showing this work's scope in adopting accelerator MCMs as NPUs in self-driving platforms (e.g., Tesla FSD).

Still, NPUs constructed as multi-chiplet modules (MCM) remain largely understudied in the context of automotive AI workloads despite the challenges and potential gains. On the one hand, automotive AI workloads exhibit unique execution flows characterized by intricate dependencies, concurrencies, and feature fusion nodes [16]–[18]. On the other, AI computing kernels exhibit varying affinities towards different accelerator types [13], [19]–[22]. Though multiple works [23]–[26] have investigated the architectural implications for improving automotive AI workloads' performance, the architectural implications of adopting chiplets technology remains to be studied.

This work aims to investigate such implications when an MCM AI accelerator is employed as the automotive NPU to accelerate AI perception workloads. Specifically, we follow the architectural template of Tesla's FSD chip [27], and simulate an industry-grade MCM inference accelerator engine – Simba [10], [15] – for the system's NPU (Figure 1). For the automotive workloads, we focus on those from the compute-intensive perception pipeline [16], [24], [28]. We implement and analyze such workloads following the Tesla Autopilot system [5] which entail HydraNets, spatio-temporal fusion, and multi-task heads (lane detection, occupancy networks). From the insights, we devise a novel scheduling methodology to enhance performance efficiency of perception workloads on the MCM-NPU. In summary, our key contributions are as follows:

- We characterize the key workloads existing in a SOTA perception pipeline (Tesla Autopilot), from the early feature extraction stage till the final multi-task heads models.
- We conduct a thorough performance breakdown of perception workloads to understand their execution properties and hardware acceleration affinities using the standard DNN performance simulator, MAESTRO [29], [30].
- From our analysis, we implement a low-cost scheduling algorithm for mapping perception workloads onto MCM-NPUs given added Network-on-Package overheads and the

978-3-9826741-0-0/DATE25/© 2025 EDAA

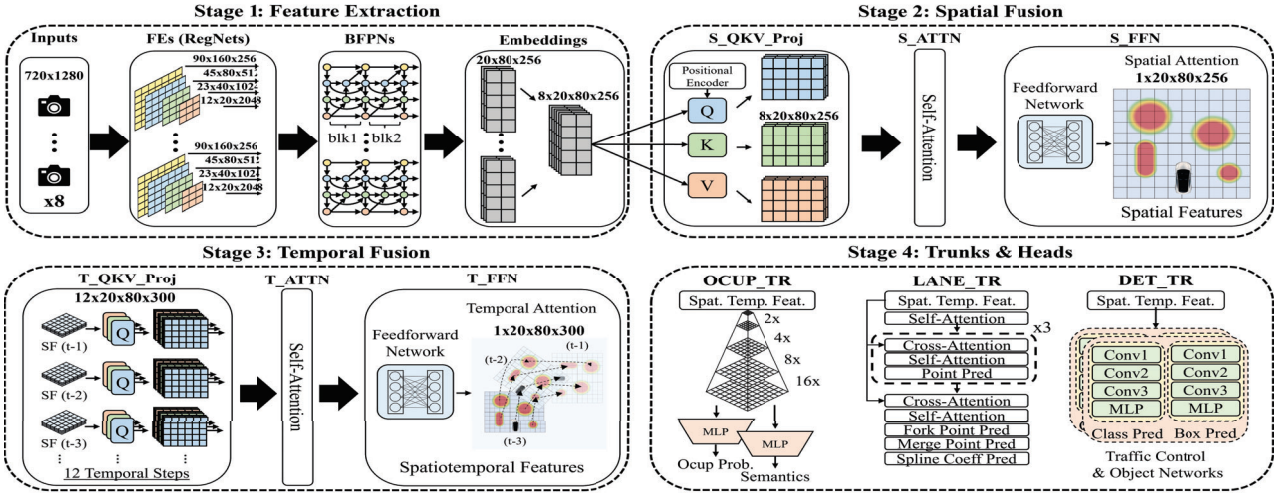

Fig. 2. The four-stage perception pipeline based on the HydraNet architecture [16] by Tesla Autopilot system [31] whose feature dims and models are displayed.

diversity of models *within* and *across* the pipeline stages.
- We evaluate our solution against existing NPU baselines showcasing performance improvement trade-offs between utilization, energy, and pipelining latency.

II. Background and Preliminaries

A. Anatomy of a self-driving platform architecture: Tesla FSD

We take the Tesla FSD (Full Self Driving) SoC as our reference self-driving architecture template. As illustrated in Figure 1, the FSD integrates the following units:

- **CPU clusters**. For general purpose compute. Each cluster constitutes a quad-core Cortex A72 in the Tesla FSD.
- **Memory**. Main memory component of the chip (e.g., LPDDR4 memory)
- **Cameras I/F and ISP**. High speed camera serial interface and image signal processor for image preprocessing
- **NPUs**. Custom-designed hardware accelerators for efficient processing of AI workloads
- **GPU**. For light-weight post-processing
- **Other**. Video encoder, safety component.

For a chiplets technology variant, a system-in-package (SiP) can be constructed from this template integrating multiple smaller chiplets, each covering a subset of components and communicating over package via high-speed interconnects. Moreover, recent prototypes like Simba [10] have shown that the NPU component can also be implemented as MCMs integrating accelerator chiplets on the package level.

B. Autonomous Driving System Pipeline

Autonomous driving systems (ADS) map sensory input data to vehicle control outputs through 3 stages: *perception* for contextual understanding and objects tracking in the driving scene; *planner* for trajectory paths generation; *control* for providing the necessary control outputs. Perception represents the most compute intensive stage [17], [24], and supports multiple driving tasks (detection, lane prediction). Recent perception modules have seen the adoption of the HydraNets architecture [16], with a *shared backbone* for extracting low-level, common features, and *specialized heads* for driving tasks specialization. In Figure 2, we illustrate the HydraNet architecture from the Tesla Autopilot system [31] discussed in the following:

Sensor inputs. Tesla Autopilot perception relies on collecting images from 8 installed cameras. Typically images can be 720p resolution at around 30 FPS [24].

Stage 1: Feature Extraction (FE+BFPN). Each raw input image is pre-processed and passed through a feature extractor (FE) followed by a bidirectional feature pyramid network (BFPN) [32], to generate multi-scale feature representations. Following [28], the FE can be a ResNet18 architecture with 4 multiscale features (90x160x256, 45x80x512, 23x40x1024, and 12x20x2048) generated throughout its ResNet blocks, which are then passed through 2 Bi-directional FPN blocks (BFPN). At the final output, multiscale features are concatenated from each FE+BFPN pipeline forming the 8x20x80x256 outputs.

Stage 2: Multi-cam Spatial Fusion (S_FUSE). Generated embeddings from each camera are fused onto a combined spatial representation in vector space (2D/3D grid map). A transformer with multi-attention head is employed, computing attention scores between every grid cell and detected features from each camera. The attention module [33] comprises: *QKV projection* to generate Query (Q), Key (K), and Value (V) vectors; *Attention* with two matrix multiplications for $(QK^T)\cdot V$; *FFN* (Feed-forward Network) comprising two linear layers. For a 20×80 2D grid [28], [31], the output becomes a fused projection of the 8 camera features onto a 1x20x80x256 grid.

Stage 3: Temporal Fusion (T_FUSE). The 1x20x80x256 spatial representation is fed into a video queue to be fused with a feature queue of N previous representations to accommodate necessary temporal features (e.g., seen signs or lane markings), and telemetry information. Another attention module can be used with N=12 for temporal fusion [28], [31], each undergoing QKV projection, attention, and FFN transformation till the final fused spatio-temporal 1x20x80x300 representation.

Stage 4: Trunks and Heads (TR). The 1x20x80x300 rep-

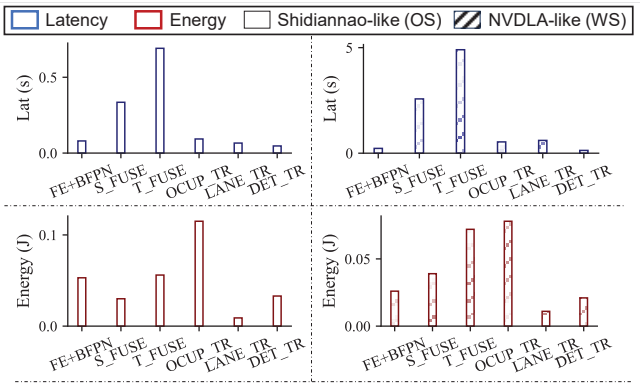

Fig. 3. Breakdown of latency (*top*) and energy consumption (*bottom*) per perception component across Shidiannao- (*left*) and NVDLA-like (*right*) accelerators using MAESTRO for a single 256 accelerator chiplet.

resentation is fed to branched trunks and heads including:

- **Occupancy network** predicting the grid's continuous occupancy probability and continuous semantics, involving 4 spatial deconvolution layers with 16x upscaling
- **Lane prediction** involving a combination of self-attention and cross-attention layers repeated 3 times for 3 levels of point predictions, and having 3 classifier predictors
- **Object detection** Multiple detection heads (traffic, vehicle, pedestrians) are supported. Each detector head entails separate class and box prediction networks using a sequence of 3 convolution layers and fully connected layer.

III. Analysis of Perception Module Workloads

To derive design insights for MCM-based NPUs, We first analyze the perception workloads' performance on various accelerator architectures from Section II-B using the analytical DNN cost model simulator, MAESTRO [29], [30] (known to have achieved 96% accuracy compared to RTL simulations). We consider the following:

- We set the number of PEs in each accelerator to 256 PEs. That way, when accelerators are assimilated as chiplets into a 6×6 MCM like Simba [10], a total of 9,216 PEs will be available equivalent to that of the original Tesla NPU [27]. We use the same operating frequency at 2 GHz [27].
- We analyze the workloads on weight stationary (WS) and output stationary (OS) accelerator types – like NVDLA [34] and Shidiannao [35]) – given their proven superiority over other accelerator types [13], [19], [36].

A. Coarse-grained Performance Analysis

In Figure 3, we breakdown the contributions of the perception stage models in terms of latency and energy on NVDLA- and Shidiannao-like accelerators and deduce the following:

OS dataflow suits latency-critical workloads. Across all workloads, the Shidiannao OS dataflow offers **6.85**× speedups over its WS counterparts, making it the prime candidate for latency critical automotive workloads [24].

WS offers energy efficiency opportunities. On average, the WS dataflows can lead to **1.2**× energy efficiency gains

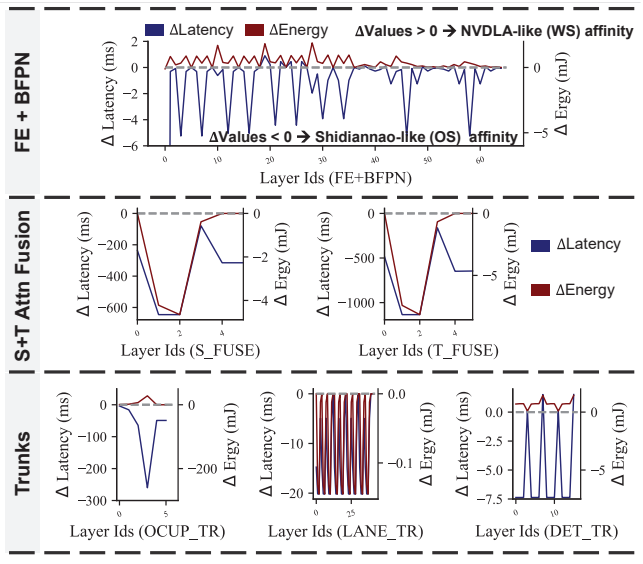

Fig. 4. Affinities of the feature extractors (*top*), spatio-temporal attention fusion *(mid)*, and trunks (bot) towards Shidiannao- and NVDLA-like accelerators. ΔValue<0 implies Shidiannao-like affinity and the opposite for NVDLA-like.

compared to OS. The gains are even more significant (**1.55**×) if we omit S_FUSE and T_FUSE from our analysis, which are more affine towards OS dataflow. This opens the door for potential heterogeneous integration (mix of OS and WS chiplets) to realize interesting performance trade-offs.

Fusion Modules are computational bottlenecks. The S_FUSE and T_FUSE modules constitute respective **25%-28%** and **52%-54%** of the overall perception module latency under both dataflow scenarios. This is attributed to the feature aggregation from multiple sources (8 cameras and 12 temporal frames) onto a shared projection space (200×80×256 grid).

FE+BFPN Scaling. Evaluations for the FE+BFPN in Figure 3 are for a single camera and to be multiplied by the 8 cameras.

B. Fine-grained Performance Analysis

We analyze the individual layers affinities towards OS and WS from the various models in Figure 4, where we compare the difference in values between the OS and WS dataflows, $\Delta Value$ = $Value_{OS}$ - $Value_{WS}$, for latency and energy (*-ve values imply OS affinity and +ve values imply WS affinity*).

FE+BFPN tradeoffs. The FE+BFPN stage exhibits a direct trade-off between latency and energy across all layers. Yet, the non-uniform distribution of performance gains, the complex dependencies of the FE and BFPN networks, and the concurrent execution requirement for 8 FE+BFPN models impose strict requirements on accelerator chiplets assignment process.

S_FUSE and T_FUSE tradeoffs. The negative evaluations of ΔLatency and ΔEnergy across all layers indicate a strong affinity towards the OS dataflow for the fusion modules. We also observe significant performance bottlenecks exist at the self-attention layers (QKV calculations at layer ids 1 and 2).

Trunks tradeoffs. The diversity of trunk models lead to varying affinities: the lane prediction trunk is completely skewed towards the OS dataflow (due to attention modules), unlike other trunks which can provide exploitable trade-offs.

C. Design Insights.

Based on our analysis, we derive the following insights onto scheduling perception workloads onto MCM-based NPUs:

- Any scheduling configuration is to target maintaining a consistent throughput (pipelining latency) across the various perception stages for streamlined input processing.
- Heterogeneous chiplets integration can be supported as long as the latency constraint is not violated. Given the dominance of the OS dataflow with regards to execution latency, we specify the pipelining latency of the OS dataflow as the reference latency constraint.
- Any optimizations for the FE+BFPN stage must be applied for the 8 concurrent models for simultaneous execution.
- The computational bottleneck of the fusion layers are confined within a small number of layers (see Figure 4), making them targets for parallelization optimizations.
- Heterogeneous chiplets integration can be particularly beneficial for the diverse trunk workloads to elevate efficiency.

IV. Proposed Scheduling Methodology

We propose to schedule the perception workloads on MCM-NPU through a throughput matching mechanism as follows:

1) Specify initial execution latency estimates and chiplet allocation per layer in each stage
2) Allocate chiplet resources across the various workload stages based on the latency estimates
3) Alleviate bottleneck stages impact via data sharding
4) Update latency estimates and chiplet assignments
5) Repeat steps 2-4 until pipelining latencies are matched or no further sharding is possible

In Algorithm 1, we achieve this sequence through a nested greedy algorithm whose outer loop alleviates bottlenecks across stages, and inner loop alleviates bottlenecks across layers in the bottleneck stages. ***Line 2*** demonstrates how chiplets are initially allocated to each stage based on a function of the model execution latencies, the number of workloads, and the number of models per each stage workload. We initially adopt uniform partitioning of chiplet resources across the 4 stages, where each is assigned a separate quadrant of chiplets. The reasons being: (i) having 4 distinctive perception stages; (ii) having 8 processing parallel model instances of the FE+BFPN; (iii) the latency bottlenecks being in the fusion modules with small number of layers; (iv) the diversity of the trunk models.

A. Choosing the FE+BFPN for the base pipelining latency

Given 8 model instances of the FE+BFPN stage where each possesing a compute latency around $\frac{1}{8}$ of the T_FUSE bottleneck (Figure 3), at least 8 chiplets need to be initially allocated to each model to maintain concurrent execution. From the latency contribution breakdown in Figure 3 across the 4 stages, assigning additional resources to the FE+BFPN would lead to suboptimal solution since it leaves just above half of the total number of chiplets for processing >90% of the overall workloads. Therefore, it becomes reasonable to specify the initial base pipelining latency to that of the FE+BFPNs models (Lat_{base}=82.7 ms) as shown in Figure 5.

Algorithm 1: Nested Greedy Throughput Matching

Inputs: W: workloads dict, **L**: latencies dict, **C**: chiplets dict, **α**: tolerance coeff
Outputs: E: execution schedule, **S***:* sharding pattern

// init pipe latencies, chiplets allocation, and base latency
1. pipe_lat_dict = $\{w_i : lat_i\}\ \forall\ lat_i \in L$
2. chiplets_map = $\{w_i : c_i\}$ s.t. $c_i = f(L, |W|, \frac{\#models}{w_i})\ \forall c_i \in C$
3. **While True:**
   // determine base pipelining latency
5. $Lat_{base} = get\_base\_pipe\_lat(W)$
6. **for** $w_i \in W$:
   // determine bottleneck workloads
7. **if** $pipe\_lat\_dict[w_i] > \alpha\ Lat_{base}$:
   // relieve layer bottlenecks
8. $l_{max} = det\_lat\_pipe\_lat(w_i)$
9. **while** $l_{max} > \alpha \cdot Lat_{base}$ **or** *used_all* **(**$chiplets\_map[w_i]$**):**
10. *shard_layer* (l_{max})
11. $l_{max} = det\_lat\_pipe\_lat(w_i)$
   // update pipe latency
12. $W[w_i] = det\_lat\_pipe\_lat(w_i)$
   // reallocate surplus chiplets
13. **if surplus_chiplets:**
14. *reallocate_chiplets_to_bottleneck*(chiplets_map)
   // update schedule and shard pattern
15. *update_schedule* (E, S)

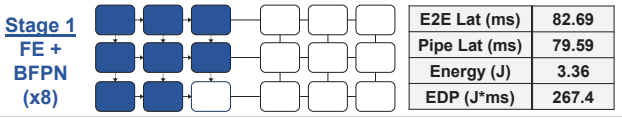

Fig. 5. The 8 FE+BFPN models mapping onto the MCM's first quadrant.

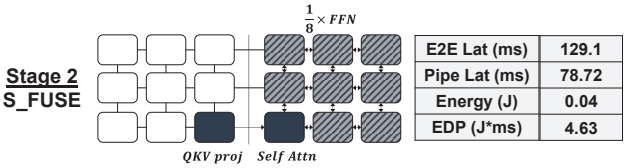

Fig. 6. Spatial Fusion (S_FUSE) mapping onto the MCM. The FFN is sharded accross 8 chiplets to maintain the pipelining latency as FE+BFPN.

B. Alleviating S_FUSE and T_FUSE bottlenecks

We alleviate the attention fusion bottlenecks through recursive sharding until the stages' latency equates that of Lat_{base}.

S_FUSE Bottleneck. This attention module operates on 8 feature input sets from the FE+BFPN stage outputs. Given 200×80×256 attention grid dimensions, S_FUSE stage incurs 78.7 ms, 20.5 ms, and 236 ms latency overheads for the QKV projection, self-attention, and Feed-Forward (FFN) layers when each is processed on an individual chiplet. Accordingly, the spatial FFN layer is sharded in a four-folded manner, replicating the FFN weights on 4 chiplets, each processing features from two FE+BFPNs. This brings FFN latency close to the QKV projection (78.7 ms) and Lat_{base} (82.7 ms). As S_FUSE is still left with 4 chiplets (one is surplus from the FE+BFPN stage), an additional sharding step can be performed, leading to the final configuration shown in Figure 6.

T_FUSE Bottleneck. Prior to sharding the 12 frames, the T_FUSE took 165.6 ms, 36.4 ms, and 490.2 ms for the QKV projection, self-attention, and FFN blocks, respectively. Our algorithm in this case involves two inner loop iterations, first distributing the FFN block workloads across 6 chiplets, leading

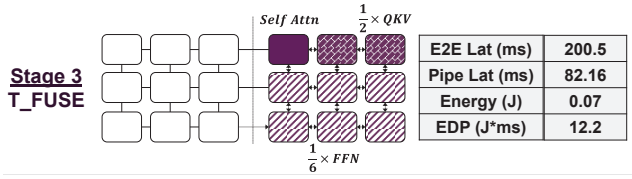

Fig. 7. Temporal Fusion (T_FUSE) mapping onto the MCM's third quadrant. QKV projection and FFN are divided across 2 and 6 chiplets, respectively.

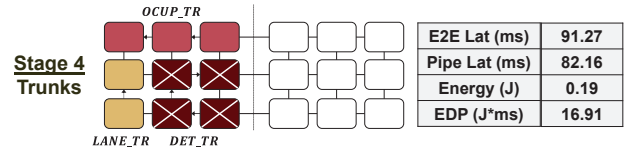

Fig. 8. Mapping of Trunks onto the fourth quadrant. Chiplets marked with the cross indicate potential placement as WS style accelerators.

FFN layer pipeline latency to drop down to 81.7 ms close to Lat_{base}. Then, the QKV projection is partitioned across two chiplets dropping the layer pipelining latency to 78.7 ms. All 9 chiplets in the quadrant are utilized as shown in Figure 7.

C. Design Space Exploration for the Trunks stage

Given the diversity of the trunk models and their customization, we employ a design space exploration for this stage to manage the mapping of workloads onto chiplets. Additionally, we consider heterogeneous integration options based on the analysis in Figure 4 to leverage potential energy efficiency gains. In particular, we consider two configurations, **Het(2)** and **Het(4)**, which integrate 2 and 4 WS chiplets within the OS-dominated 3×3 chiplets quadrant, respectively. We specify the following scoring function for our search:

$$Score(config) = \begin{cases} -\infty, & \text{if } \exists L_{max}(\text{chiplet}) > L_{cstr} \\ -1 \times EDP, & otherwise \end{cases}$$

where a scheduling configuration ($config$) is only considered on the basis of its Energy-Delay Product evaluation as long as the pipeline latency (L_{cstr}) is not violated by any chiplet. Given the relatively small size of the search space (9 chiplets, 3 models, and $<$ 100 layers), we perform a brute force search with the results provided in Table I at L_{cstr} **= 85 ms**.

From the table, we can see that the heterogeneous configurations, **Het(2)** and **Het(4)** realize performance efficiency improvements in energy and EDP compared to the **OS** only configuration, **achieving average 1.1% and 6.2% reductions in raw energy; 17.4% and 12.0% reductions in EDP, respectively.** Upon careful inspection, we found the WS chiplets are predominantly assigned to the DET_TR layers, leading DET_TR independently to achieve a 35% reduction in energy.

D. NoP Cost Modeling

We model the NoP data movement overheads using a cost model built around MAESTRO [29], [30] and the microarchitecture parameters from Simba [10] scaled to 28 nm as follows:

- NoP Interconnect BW: 100 GB/s/chiplet
- NoP Interconnect Lat: 35 ns/hop
- NoP Interconnect Ergy: 2.04 pJ/bit

TABLE I
RESULTS OF HETEROGENEOUS INTEGRATION FOR THE MCM TRUNKS RELATIVE TO THE OS ONLY CONFIGURATION. L_{cstr} IS SET TO 85 MS. Δ MEANS THE % CHANGE BETWEEN HETERO. AND OS CONFIGURATIONS.

Metric	OS	WS	Het(2)	Het(4)	Δ(2)	Δ(4)
E2E Lat(ms)	91.2	605.7	91.3	91.3	+0.1%	+0.1%
Pipe Lat(ms)	87.9	605.7	71.7	71.7	-18.4%	-18.4%
Energy(J)	0.185	0.139	0.183	0.174	**-1.1%**	**-6.2%**
EDP(ms*J)	16.89	59.35	14.38	15.1	**-17.4 %**	**-12.0%**

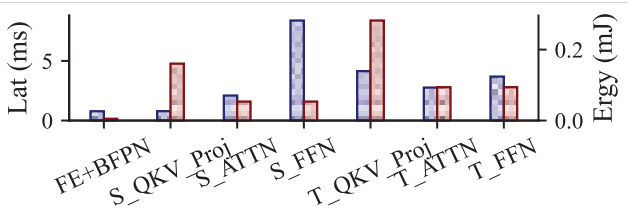

Fig. 9. The NoP data movement costs in terms of latency (*blue*) and energy (*maroon*) throughout the first 3 stages of the perception pipeline.

Transmission latency is computed as a function of feature map size over the NoP bandwidth, multiplied by the number of hops from source to destination. Whereas transmission energy is the multiplication of feature map size, interconnect bit transmission energy, and the number of hops.

We analyze the NoP data movement overheads in Figure 9 across the different layer workloads following the scheduling configuration of the previous subsections and make the following observations: (i) Large feature map outputs (as from QKV_Proj layers) have high transmission costs (latency and energy) and are placed in close proximity to their destinations to limit overheads (Figures 6 and 7); (ii) The gathering of sharded outputs from multiple far nodes (S_FFN) can cause rise in NoP traffic and the corresponding latency. (iii) Most importantly, the overall perception latency bottleneck does not lie in NoP transmission overheads as they are at least two orders of magnitude less than the computational costs in Figure 3.

V. EVALUATION

We evaluate our MCM scheduling solution against baselines with the same number of PEs:

- A single accelerator chiplet with 9,216 PEs (Regular)
- Two accelerator chiplets with 4,608 PEs each
- Four accelerator chiplets with 2,304 PEs each

We consider two pipelining schemes for the evaluation: (1) **stagewise**, and (2) **layerwise**. We also analyze how our solution can scale to two NPUs (2×6×6 Simba MCMs for a total of 72 chiplets). Unless otherwise stated, we focus our analysis on the multi-chiplet NPU with OS only dataflow. We perform comparisons on the first 3 bottleneck stages of the perception pipeline. A separate ablation study is performed for the trunks.

A. Comparison against Baselines

We show the comparison against the baselines in Table II. Though the 6×6 solution achieves the best pipelining latency scores with 87 ms, it incurs additional energy consumption from the overhead associated with the NoP data transmission, leading it to incur a 10.9% increase in energy consumption

TABLE II
PERFORMANCE EVALUATION FOR VARIOUS CHIPLET ARRANGEMENTS FOR THE SAME TOTAL NUMBER OF PES (9,216). 36×256 IS SIMBA-LIKE [10]

Pipeline	Metric	1x9216	2x4608	4x2304	36x256
Stagewise	**E2E Lat(s)**	1.8	1.8	1.8	0.5
	Pipe Lat(s)	1.8	0.7	0.67	0.09
	Energy(J)	0.64	0.69	0.65	0.71
	EDP(ms*J)	274	283	273	69
	Utilization(%)	19.11	25.39	31.13	54.19
Layerwise	**E2E Lat(ms)**	1.8	1.5	1.3	0.5
	Pipe Lat(ms)	1.8	0.5	0.5	0.09
	Energy(J)	0.64	0.64	0.65	0.71
	EDP(ms*J)	274	141	86	69
	Utilization(%)	19.11	26.20	31.86	54.19

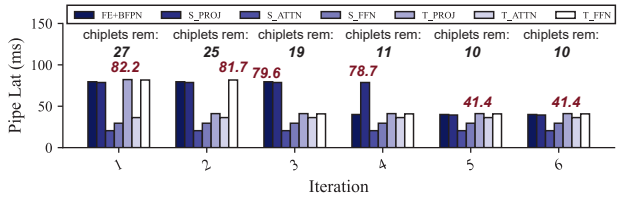

Fig. 10. Algorithm 1 steps when alleviating pipelining latency when two Simba NPUs are active. Numbers in (*red*) indicate corresponding pipelining latency.

compared to the single chiplet solution. Still, the 6×6 solution achieves the lowest EDP of 69 J*ms as a result of low end-to-end latency from the cross-chiplet sharding. In terms of PEs utilization, the 6×6 solution achieves an average of 54.19% utilization across all chiplets' PEs, constituting a 2.8×, 2.1×, and 1.7× increase over the three respective baselines.

B. Scaling to 2 Multi-chiplet NPUs

We explore how our solution scales to 72 chiplets if the two NPUs on the FSD are active and processing the same workloads (recall Fig 1). We assume in this analysis the number of trunks is doubled (assigned 2×9 chiplets), and that they incur a fixed performance overhead not becoming the the latency bottleneck.

In Figure 10, we show how our algorithm progresses to reduce the pipelining latency and assigns chiplet resources. First, our algorithm extends the sharding of the temporal projection layer T_QKV from 2 to 4, reducing its pipelining latency to 41.1 ms. Then, T_FFN layer is assigned an additional 6 chiplets to bring its latency down to 40.8 ms. At this point, the sharding is exhausted for the T_FFN as each temporal frame is processed independently on a separate chiplet. The FE+BFPN layer is then partitioned into two pipelining stages at the fourth convolutional ResNet-18 block, yielding two equivalent partitions with pipelining latency of 40.04 ms. Lastly, S_PROJ is divided across two chiplets to yield a latency at 39.4 ms. The final pipelining latency is 41.1 ms, almost 2× that of the 36 chiplet case.

C. Ablation on Optimizing Trunk models Design

We performed additional ablation studies on the trunk models to pinpoint their performance bottlenecks.

TABLE III
ANALYSIS ON THE INPUT SCALING EFFECTS ON LATENCY FOR OCUP_TR

Upsampling Factor	[2X,2Y]	[4X,4Y]	[8X,8Y]	[16X,16Y]
E2E Lat(ms)	0.97	4.97 (4.10x↑)	21.16 (20.72x↑)	86.29 (87.59x↑)
Pipe Lat(ms)	0.97	3.99 (3.11x↑)	16.18 (15.64x↑)	65.13 (66.00x↑)

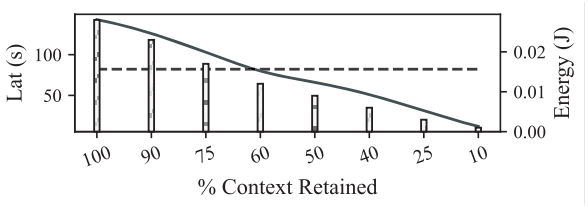

Fig. 11. Lane trunk latency (*lineplot*) and energy (*barplot*) under context-aware computing. Dashed line indicates pipelining latency threshold at 82 ms.

Occupancy Trunk. Deconvolution operations are the bottleneck of the occupancy trunk. In Table III, we showcase the non-linear increase in latency incurred in the occupancy network with each added upsampling layer, with the final upsampling layer contributing the most to the overall latency (∼75%). This underpins an exploitable trade-off between resolution granularity and performance efficiency.

Lane Prediction Trunk. In the Tesla Autopilot System, the lane prediction network adopts a form of context-aware computing for efficiency, where rather than running the lane prediction algorithm for every region, processing is only performed for relevant regions in the grid [37]. In Figure 11, we show a similar pattern in our implementation where processing across all regions leads latency to exceed the 82 ms pipelining latency. Around 60% computing satisfies the latency constraint.

VI. RELATED WORKS

A. Multi-chiplet Modules

Accelerating AI workloads through a chiplets system involves combining on package a host system with an acceleration platform which can be based on GPU [38], [39], FPGA [40], or NPUs [10], [11], [15]. More sophisticated architectures can entail the acceleration platform comprising multiple smaller modules consolidated on package to form a multi-chiplet module (MCM) [10]–[13], [15], [38].

B. Autonomous Driving Systems Efficiency

The works in [17], [26], [41], [42] study how distributed scheduling of ADS perception tasks across multiple platforms elevates efficiency. Other works in [23], [24] explore how different hardware architectures and platforms (GPUs, FPGAs, ASICs) influence efficiency in the autonomous driving SoC.

VII. CONCLUSION

We have studied the performance implications from adopting chiplet-based NPUs to accelerate perception AI workloads through the Tesla Autopilot use case. Our findings demonstrate that the combination of throughput matching algorithm and heterogeneous integration offer desirable performance trade-offs from the multi-chiplet module despite the added NoP costs, motivating further studies on adopting forthcoming chiplet-based NPU architectures in the automotive setting.

REFERENCES

[1] Bosch, "E/E architecture," https://www.bosch-mobility.com/en/mobility-topics/ee-architecture/, 2023.

[2] Boston Consulting Group, "The future of automotive compute," https://media-publications.bcg.com/The-Future-of-Automotive-Compute.pdf, 2023.

[3] Fraunhofer IIS and Siemens EDA, "Chiplets - a platform-concept for automotive adas systems," https://chipletsummit.com/proceeding_files/a0q5f000001WuE0/20230125_A-102_Heinig.PDF, 2023.

[4] imec, "Why are chiplets attracting the attention of the automotive industry?" https://www.imec-int.com/en/articles/why-are-chiplets-attracting-attention-automotive-industry, 2023.

[5] Towards AI, "Tesla's self driving algorithm explained," https://towardsai.net/p/l/teslas-self-driving-algorithm-explained, 2022.

[6] Fierce Electronics, "GM Cruise designs custom chips at lower cost for self-driving vehicles such as Origin," https://www.fierceelectronics.com/sensors/gm-cruise-designs-custom-chips-lower-cost-self-driving-vehicles-such-origin, 2022.

[7] Sunrise Volkswagen, "Volkswagen Plans on Designing Chips for Its Self-Driving Cars," https://www.sunrise-vw.com/volkswagen-plans-on-designing-chips-for-its-self-driving-cars/, 2021.

[8] Marvell, "Marvell joins imec automotive chiplet initiative to facilitate compute socs for super-human sensing," https://www.marvell.com/blogs/marvell-joins-imec-automotive-chiplet-initiative-to-facilitate-compute-socs-for-super-human-sensing.html, 2023.

[9] Synopsys, "UCIe Standard: Benefits & Requirements Explained," https://www.synopsys.com/blogs/chip-design/ucie-standard-multi-die-systems.html, 2023.

[10] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[11] J. Cai, Z. Wu, S. Peng, Y. Wei, Z. Tan, G. Shi, M. Gao, and K. Ma, "Gemini: Mapping and architecture co-exploration for large-scale dnn chiplet accelerators," *arXiv preprint arXiv:2312.16436*, 2023.

[12] Z. Tan, H. Cai, R. Dong, and K. Ma, "NN-baton: DNN workload orchestration and chiplet granularity exploration for multichip accelerators," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1013–1026.

[13] M. Odema, L. Chen, H. Kwon, and M. A. A. Faruque, "SCAR: Scheduling Multi-Model AI Workloads on Heterogeneous Multi-Chiplet Module Accelerators," *arXiv preprint arXiv:2405.00790*, 2024.

[14] Z. Wang, G. R. Nair, G. Krishnan, S. K. Mandal, N. Cherian, J.-S. Seo, C. Chakrabarti, U. Y. Ogras, and Y. Cao, "AI computing in light of 2.5 d interconnect roadmap: Big-little chiplets for in-memory acceleration," in *2022 International Electron Devices Meeting (IEDM)*. IEEE, 2022, pp. 23–6.

[15] B. Zimmer, R. Venkatesan, Y. S. Shao, J. Clemons, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "A 0.32–128 tops, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 4, pp. 920–932, 2020.

[16] R. T. Mullapudi, W. R. Mark, N. Shazeer, and K. Fatahalian, "Hydranets: Specialized dynamic architectures for efficient inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8080–8089.

[17] A. Malawade, M. Odema, S. Lajeunesse-DeGroot, and M. A. Al Faruque, "SAGE: A split-architecture methodology for efficient end-to-end autonomous vehicle control," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–22, 2021.

[18] A. V. Malawade, T. Mortlock, and M. A. Al Faruque, "Hydrafusion: Context-aware selective sensor fusion for robust and efficient autonomous vehicle perception," in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2022, pp. 68–79.

[19] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-dnn workloads," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 71–83.

[20] I. Panopoulos, S. Venieris, and I. Venieris, "Carin: Constraint-aware and responsive inference on heterogeneous devices for single-and multi-dnn workloads," *ACM Transactions on Embedded Computing Systems*, 2024.

[21] H. Bouzidi, M. Odema, H. Ouarnoughi, S. Niar, and M. A. Al Faruque, "Map-and-conquer: Energy-efficient mapping of dynamic neural nets onto heterogeneous mpsocs," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[22] X. Zhang, C. Hao, P. Zhou, A. Jones, and J. Hu, "H2h: heterogeneous model to heterogeneous system mapping with computation and communication awareness," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 601–606.

[23] S. Liu, J. Tang, Z. Zhang, and J.-L. Gaudiot, "Computer architectures for autonomous driving," *Computer*, vol. 50, no. 8, pp. 18–25, 2017.

[24] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 751–766.

[25] S. Baidya, Y.-J. Ku, H. Zhao, J. Zhao, and S. Dey, "Vehicular and edge computing for emerging connected and autonomous vehicle applications," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[26] L. Chen, M. Odema, and M. A. A. Faruque, "Romanus: Robust task offloading in modular multi-sensor autonomous driving systems," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–8.

[27] E. Talpes, D. D. Sarma, G. Venkataramanan, P. Bannon, B. McGee, B. Floering, A. Jalote, C. Hsiong, S. Arora, A. Gorti *et al.*, "Compute solution for tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020.

[28] Tesla. Tesla AI Day 2021. Youtube. [Online]. Available: https://www.youtube.com/watch?v=j0z4FweCy4M&ab_channel=Tesla

[29] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings," *IEEE micro*, vol. 40, no. 3, pp. 20–29, 2020.

[30] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 754–768.

[31] Think Autonomous, "How Tesla's Autopilot Works," https://www.thinkautonomous.ai/blog/how-tesla-autopilot-works/, 2021.

[32] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 781–10 790.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[34] NVIDIA, "Nvdla deep learning accelerator," http://nvdla.org,2017., 2023.

[35] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 92–104.

[36] S. Kim, H. Kwon, J. Song, J. Jo, Y.-H. Chen, L. Lai, and V. Chandra, "DREAM: A Dynamic Scheduler for Dynamic Real-time Multi-model ML Workloads," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, 2023, pp. 73–86.

[37] Tesla. Tesla ai day 2022. Youtube. [Online]. Available: https://www.youtube.com/watch?v=ODSJsviD_SU&t=3711s&ab_channel=Tesla

[38] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-chip-module GPUs for continued performance scalability," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 320–332, 2017.

[39] "Nvidia Grace Hopper Superchip," https://www.nvidia.com/en-us/data-center/grace-hopper-superchip/, 2024.

[40] R. Hwang, T. Kim, Y. Kwon, and M. Rhu, "Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 968–981.

[41] M. Odema, L. Chen, M. Levorato, and M. A. Al Faruque, "Testudo: Collaborative intelligence for latency-critical autonomous systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 6, pp. 1770–1783, 2022.

[42] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.