Work-in-Progress: Evaluating Stage-Prioritized IDK Cascades in Embedded Real Time Systems

Comyar Dehlavi
Memorial High School
Houston, TX, 77024, USA
dehlavi.comyar@outlook.com

Albert M. K. Cheng
Department of Computer Science
University of Houston
Houston, TX, 77004, USA

Thomas Carroll
Department of Computer Science
University of Houston
Houston, TX, 77004, USA

***Abstract*—IDK (I Don't Know) cascades are classification frameworks that selectively switch between models based on confidence thresholds; when confidence is insufficient for a model, a more complex model is iterated to up to a deterministic classifier. This paper proposes and evaluates a stage-prioritized variant of the IDK cascade, in which constituent neural networks are trained and selected for performance in specific phases of system behavior. The system was implemented on a Raspberry Pi Zero W to regulate the temperature of a Peltier element at a constant 16°C. Initial trials were ran to do this utilizing the PID (proportional, integral, derivative) control algorithm, a well established algorithm used in embedded systems for getting a linear output proportional to the change in some input. Two neural networks each with two layers were trained using the PID-derived trial data, a standard derived trial data in order to optimize the models with the goals of lowering latency, reducing power usage, and lowering the standard deviation of the temperature readings. The first, faster network (13 neurons) was prioritized during the early cooling phase when the temperature was farthest from the setpoint; the second (49 neurons), more accurate network was favored near steady-state conditions. Results show that this stage-aware cascade outperformed both individual neural networks and the deterministic PID fallback in terms of both temperature stability and power efficiency at high enough confidence thresholds.**

I. Introduction

In embedded real time systems, striking a balance between lowering computational cost and increasing accuracy of the system is often a major challenge for developers. Factors like power consumption, execution time, and efficacy of the system's output all determine the overall desirability of the system. This often leads to a dichotomy between engineers and developers as to whether they should prioritize well established algorithms or resort to more modern and advanced models such as neural networks for control systems. In recent years, new research has presented IDK (I Don't Know) classifiers which dynamically switch between deep learning models based on a set confidence threshold, in which the system resorts to a reliable deterministic fallback algorithm when the confidence is below the threshold [1]. IDK cascades accomplish the goal of balancing low execution times and relatively high accuracy, making it desirable for embedded systems where both must be accomplished for overall performance of the system [3].

The concept of IDK Cascades as confidence-based classifiers was introduced by Sanjoy Baruah (2023) for switching based on probabilistic scheduling and having fallback mechanisms which are deadline-aware [1]. While this in theory led to better average latency and performance [2], in the context of embedded systems it was agnostic to the stage of runtime. This paper builds upon that foundation by implementing a version of the IDK cascade that prioritizes certain models based on the portion of the program in which it is currently in, as well as having each neural network be more so targeted for specific parts of the program in addition to also considering hardware limitations and factors like power consumption.

Despite the benefits of IDK cascades in embedded systems, they often are underdeveloped in the way they determine which classifier may be best suited depending on the specific stage or segment of the application's runtime. This issue stems from the iterative nature of most IDK cascades, in which they only switch classifiers and resort to other models with low confidence being the only factor. In embedded systems with a clear runtime course, having the IDK cascade inherently recognize the strengths of each model, giving preference depending on the stage, can greatly improve and strengthen the applications overall runtime performance.

When designing an IDK cascade for embedded systems, not only can stage aware processing improve performance [2], but giving individual neural networks awareness of the important factors like system latency and power consumption in their training can work to greatly strengthen the balance of performance-accuracy which embedded developers are actively seeking. Individual networks can then be further refined and specified for their portion of the applications runtime by varying the amount of neurons to control their speed and responsiveness. This allows for networks to be better adjusted for critical portions of the application.

Our Main contributions in this work include,We introduce a holistic approach to IDK cascades of two neural networks along and a PID deterministic fallback, with each simple neural network trained to focus on a certain type of scenario (input). We evaluate our model using a Raspberry Pi Zero W to maintain a regulatory temperature of a Peltier element at setpoint 16°C. We leverage the reduced execution time to create an IDK cascade with higher accuracy and lower power usage than that of the deterministic PID model.

II. Methodology

A. IDK Cascade Logic

We designed our stage-prioritized around the aforementioned Peltier element. The controller uses an IDK cascade of three control models: a fast but less accurate neural network (NN1), a slower but more accurate neural network (NN2), and the PID controller [3]. At each time step, the controller evaluates the confidence of the predicted duty cycle from a model. If the confidence falls below a fixed threshold, the controller iterates to the next model, all while having a deterministic fallback, usually the PID controller.

Unlike conventional IDK implementations, which use only confidence to determine switching, this cascade also incorporates a stage-prioritization strategy. NN1 is given preference in it's confidence reading during the early cooling stage when the system's temperature is more than 4°C from the setpoint due to its more aggressive control and speed, while similarly NN2's confidence reading is given a priority boost when the temperature is within 4°C of the setpoint, in which it exhibits greater stability. The PID controller is invoked when the confidence of both networks is below the chosen threshold or ineffective beyond the neural network's capacities.

We also incorporated hysteresis margins, dwell counters, and a confidence history mechanism into the IDK cascade. These were included to reduce controller oscillation and ensure stable transitions between models. There was also a deadline of 1.5 seconds so that the models could maintain real-time responsiveness.

B. Neural Network Design and Training

The two neural networks were trained using TensorFlow and implemented on the Raspberry Pi Zero W using TensorFlow Lite Micro due to hardware restraints. Both networks have two hidden layers, specified:

- **NN1 (Fast Network):** 8 neurons in the first layer, 4 in the second. Total: 13 neurons.
- **NN2 (Slow Network):** 32 neurons in the first layer, 16 in the second. Total: 49 neurons.

The fewer number of total neurons in NN1 give it faster processing speed, but also leads to one of its major attributes being more aggressive cooling and outputting a higher duty cycle, which is preferred when the temperature difference between measured and setpoint is largest. Since NN2 has much more neurons, it is thus slower, but more precise and accurate, making it better suited for small fluctuations and when the measured temperature is hovering around the setpoint.

Each network receives three input features: current temperature, current power level, and estimated latency of the previous control step and outputs a predicted PWM duty cycle from 0-100. Both models were trained on labeled datasets generated from standalone PID trials, in which the system was controlled using a PID algorithm and data was logged every second for a total of 45 minutes. The objective during training was to minimize a weighted loss function prioritizing low latency, low power consumption, and minimal standard deviation in temperature.

C. Stage-Prioritization Strategy

The IDK-Cascade implementation features stage-aware prioritization of classifiers. Unlike traditional IDK cascades by Sanjoy Baruah et al. [1], which select models based solely on confidence scores, this approach prioritizes models based on the difference between recorded and desired temperatures, increasing their selection likelihood. While retaining the core concept of switching to the next model when confidence falls below a threshold for optimal execution time, this implementation adds nuance by incorporating additional factors for model preference in the stage-prioritization strategy.

When the recorded temperature significantly exceeds the target of 16°C, the controller prioritizes NN1 for its responsiveness. As the temperature nears the target (within ±4°C) and cooling demand decreases, NN2 is favored for its finer control. The PID controller is used as a safety and recovery mechanism, activating during unexpected confidence drops, persistent oscillations, or failure to approach the baseline temperature.

Algorithm 1 Stage-Prioritized IDK Cascade Control Loop

Measure T, update dwell, and get threshold θ
for controller $\in$ {NN_FAST, NN_SLOW} **do**
  Compute confidence c
  if $|T - T_{set}| \leq 4$ **then** c prefers NN_SLOW
  else if $|T - T_{set}| > 4$ **then** c prefers NN_FAST
  end if
  if $c \geq \theta + 0.1$ **and** dwell ≥ 6 **then**
    Switch to this controller ▷ hysteresis margin & dwell counter
    break
  else if controller = prior and $|c - c_{prev}| < 0.2$ **then**
    Stay with same controller
    break
  end if
end for
if controller chosen **then** $u \leftarrow$ controller output
else $u \leftarrow$ PID output
end if
Apply duty u via PWM

D. System Architecture

A Raspberry Pi Zero W was used for implementing the thermal regulation system, which controlled a Peltier thermoelectric cooling element using PWM (Pulse-width modulation). Four TMP36 temperature sensors were attached to the surface of the element, and an average reading was fed into the control algorithm at a fixed sampling interval. The controller's output is a duty cycle between 0 and 100, which determines the intensity of the Peltier cooling via PWM. The system is powered through a regulated 5V supply, and temperature feedback is used to maintain a baseline of 16°C. A

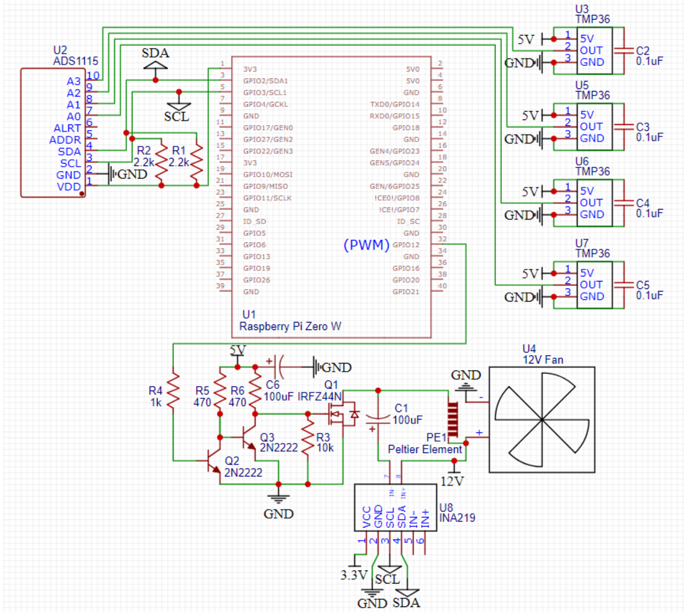

Fig. 1. Circuit schematic of the thermal regulation system using Raspberry Pi Zero W, temperature sensors, and Peltier element.

schematic of the control circuit is shown in Fig. 1. Additional documentation can be found at this paper's GitHub repository.[1]

III. Experiments and Results

A. Experimental Setup

Each trial began with the Peltier surface at 26°C, ran for approximately 45 minutes, and maintained a 16°C baseline. The Peltier element was powered by a Raspberry Pi Zero W, which used PWM (Pulse Width Modulation) to control cooling. PWM rapidly toggles the output pin connected to the element, with the duty cycle (0-100% on/off) determining cooling intensity. Four configurations were tested: PID-only, NN1-only, NN2-only, and IDK Cascade. The IDK Cascade was further evaluated at 30%, 50%, and 70% confidence intervals, following Sanjoy Baruah's logic [2]. If the neural network's confidence in generating the PWM duty cycle fell below the trial's threshold, it switched to the next network, bypassing the novel stage-prioritization strategy.

In each trial we recorded the average power consumption, average latency (execution time), how accurate the temperature reading were, and the amount of time each controller was used within the cascade.

B. Controller Performance Comparison

Table 1 displays the average power consumption, latency, and accuracy relative to the baseline of each configuration. The PID controller was found to be the fastest model in terms of computation speed in comparison to the others at just 3.430 ms as well as a relatively high accuracy of 97.39%. When comparing the standalone neural networks, NN2 was much more accurate than NN1, with 94.781% accuracy in comparison to 83.32%, which is consistent with the training implementation as the majority of the runtime is when the measured temperature is hovering around the baseline, with NN1 being better at the initial phase and NN2 under performing initially. NN1 has the highest power consumption at 18.899 W, which is also consistent with the notion of it being a more aggressive coolant and model.

However, when combining the models into IDK cascades, we see that at 50% confidence the cascade performs nearly as well as the fallback PID algorithm, and at 70% the IDK cascade manages to outperform all other models, including the PID, with 98% accuracy and 14 W power consumption. These results suggest that blending NN1's speed and NN2's precision through dynamic switching at high enough confidence intervals leads to superior performance.

TABLE I
Comparison of Average Power Usage, Latency, and Temperature Stability Across Control Strategies.

Model	Accuracy (%)	Latency (ms)	Power (W)
PID	97.39	3.430	16.396
NN1	83.32	12.569	18.899
NN2	94.71	12.708	17.324
IDK (30%)	96.44	13.776	15.722
IDK (50%)	97.27	13.713	17.208
IDK (70%)	98.48	13.378	14.225

C. Stage Activation Breakdown

Figure 2 shows controller usage distribution within the IDK Cascade at different confidence thresholds. At 30%, we see NN2 being significantly dominant over NN1 and NN2 due to its inherent preference by the cascade since it's designed stage makes up most of the runtime, but also because it has a very wide ranger of operation with due to the low confidence interval. At 50% confidence we see a slightly lower portion of the program being dominated by NN2, but we also see the PID and NN1 controller be active about the same percentage of the time. At 70% confidence, which is when we notice the cascade start to outperform all other metrics, we notice that the PID is being executed the vast majority of the runtime, while NN2 is only slightly higher than NN1. This is consistent with the confidence threshold being high as the networks are more limited and more often output "I don't know."

The stage-prioritization implemented alongside the traditional IDK cascade framework leads to a more predictive distribution of models and allows the developer to exhibit greater control on how the cascade specifically interacts and influences the output and model selection.

D. Temperature Regulation Over Time

Figure 3 depicts the temperature of each control model vs time over a major portion of each 45 minute trial. A significant finding can be that NN1 cools to a much greater extent compared to the other models and much faster; however, it overshoots past the 16°C baseline and continued to overshoot, being approximately 12.5°C towards the end of the trial. In the NN2 trial, significant oscillations can be observed in comparison to the IDK cascade models, along with a recognizable amount of oscillation towards the middle of the trial for the IDK cascade at a 30% confidence threshold. The PID, 50%

[1] All source code, trained models, trial data, and documentation are available at https://github.com/Void-Overflow/Neural-Network-vs-PID-in-IDK-Cascade-for-Thermal-Regulation-Research

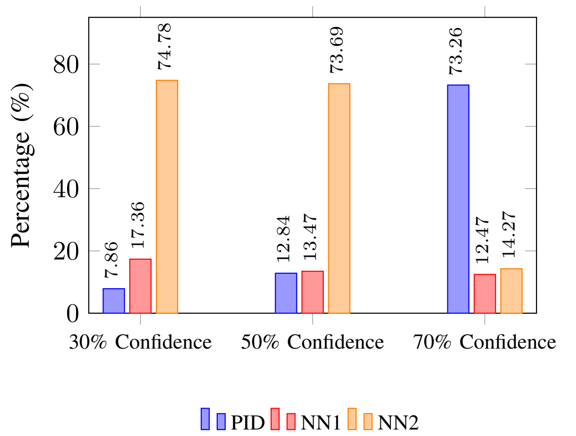

Fig. 2. Proportion of controller usage at varying IDK confidence thresholds.

IDK, and 70% IDK experienced minimal osculations and overall tended to stick much better to the 16°C baseline.

These traces demonstrate that combining these models and focusing on their strengths during certain portions in moderation leads ultimately to the most stable possible execution in terms of thermo-efficiency.

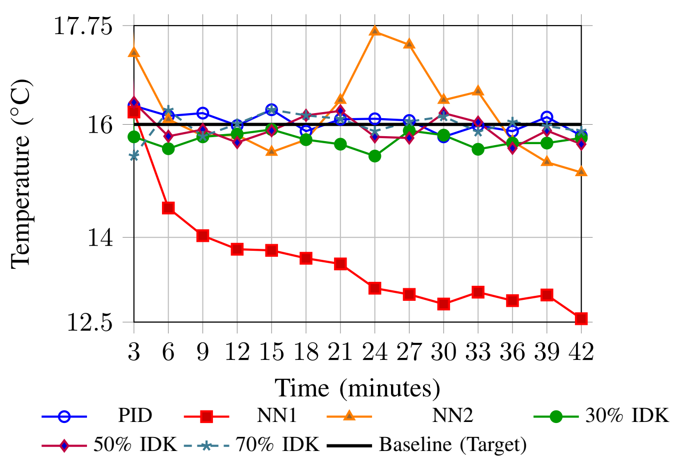

Fig. 3. Temperature readings for different controllers with baseline at 16°C.

IV. Discussion

The results of this publication display strong results that the stage-aware implementation of the IDK cascade can substantially improve the performance of embedded systems. Traditional IDK models make switching decisions based purely off model confidence, which isn't as efficient in the context of embedded systems with additional hardware and runtime limitations which are needed to be considered [3]. By distinguishing certain neural networks for certain predictable phases of the runtime, and allowing a fallback algorithm in times of uncertainty, this allows the IDK cascade to be much more precise and not only more accurate in its output, but also more efficient in terms of using limited computational resources.

The performance gains seen in this experiment are multi-faceted. The stage-aware IDK cascade at 70% confidence not only managed to increase the accuracy and therefore stability of the trial to be higher than that of the fallback, but it also reduced power consumption when compared to any of the other models. This highly suggests that utilizing preference for certain networks based on the segment of runtime which it is in, as well as specifically training the networks to be aware of hardware limitations, directly addresses the issue of balancing responsiveness and stability with energy efficiency in embedded environments. Furthermore, the application of hysteresis and threshold tuning helped prevent oscillations between models, reduced jitter, and help smooth and make transitions between models more smooth.

Compared to the IDK cascade framework presented by Baruah et al. [1], the implementation presented in this paper introduced a more deliberate selection mechanism that implemented greater awareness of runtime and hardware limitations in systems where computational power is finite rather than just focusing on confidence scores alone. While the analysis presented in that work focused on optimizing switching primarily on minimizing execution deadlines [2], this paper expands on that work by considering factors that are additionally significant in embedded systems, especially when tied to physical processes like thermo-regulation.

V. Conclusion

The findings of this paper present a further enhanced version of the IDK cascade specifically tailored to the environment of embedded real time systems, in which it combined runtime stage-awareness with the traditional switching associated with IDK classifiers. In the case study of thermal regulation with a Raspberry Pi Zero W and Peltier element, the stage-aware cascade ultimately proved more effective compared to both traditional PID control and standalone neural networks. It had tighter temperature control, lower power usage, and ultimately more control over the runtime.

Ultimately, this work contributes to the growing field of hybrid intelligent-deterministic systems. The need for the dynamic, precise, yet efficient stage-aware control model presented in this study is especially relevant in this day-and-age where applications and innovations are using increasingly complex and demanding learning-based models. The stage-prioritized IDK cascade demonstrates that its implementation into systems can ultimately yield meaningful gains in precision and efficiency-insights that could benefit a wide variety of embedded applications in the future.

References

[1] S. Baruah, A. Burns, R. I. Davis, and Y. Wu, "Optimally ordering IDK classifiers subject to deadlines," *Real-Time Systems*, vol. 59, no. 1, pp. 1–34, 2023. doi: https://doi.org/10.1007/s11241-022-09383-w.

[2] S. Baruah, T. Abdelzaher, K. Agrawal, A. Burns, R. I. Davis, Z. Guo, and Y. Hu, "Scheduling IDK classifiers with arbitrary dependences to minimize the expected time to successful classification," *Real-Time Systems*, vol. 59, pp. 348–407, 2023. doi: https://doi.org/10.1007/s11241-023-09395-0.

[3] X. Wang, Y. Luo, D. Crankshaw, A. Tumanov, F. Yu, and J. E. Gonzalez, "IDK cascades: Fast deep learning by learning not to overthink," arXiv.org, 2018. https://arxiv.org/abs/1706.00885.