

# QSLM: A Performance- and Memory-aware Quantization Framework with Tiered Search Strategy for Spike-driven Language Models

Rachmad Vidya Wicaksana Putra, Pasindu Wickramasinghe, Muhammad Shafique  
*eBRAIN Lab, New York University (NYU) Abu Dhabi, Abu Dhabi, United Arab Emirates*  
 {rachmad.putra, pmw6287, muhammad.shafique}@nyu.edu

**Abstract**—Large Language Models (LLMs) have been emerging as prominent AI models for solving many natural language tasks due to their high performance (e.g., accuracy) and capabilities in generating high-quality responses to the given inputs. However, their large computational cost, huge memory footprints, and high processing power/energy make it challenging for their embedded deployments. Amid several tinyLLMs, recent works have proposed spike-driven language models (SLMs) for significantly reducing the processing power/energy of LLMs. However, their memory footprints still remain too large for low-cost and resource-constrained embedded devices. Manual quantization approach may effectively compress SLM memory footprints, but it requires a huge design time and compute power to find the quantization setting for each network, hence making this approach not-scalable for handling different networks, performance requirements, and memory budgets. To bridge this gap, we propose *QSLM*, a novel framework that performs automated quantization for compressing pre-trained SLMs, while meeting the performance and memory constraints. To achieve this, QSLM first identifies the hierarchy of the given network architecture and the sensitivity of network layers under quantization, then employs a tiered quantization strategy (e.g., global-, block-, and module-level quantization) while leveraging a multi-objective performance-and-memory trade-off function to select the final quantization setting. Experimental results indicate that our QSLM reduces memory footprint by up to 86.5%, reduces power consumption by up to 20%, maintains high performance across different tasks (i.e., by up to 84.4% accuracy of sentiment classification on the SST-2 dataset and perplexity score of 23.2 for text generation on the WikiText-2 dataset) close to the original non-quantized model while meeting the performance and memory constraints. Hence, QSLM framework advances the efforts in enabling efficient design automation for embedded implementation of SLMs.

**Index Terms**—Large Language Models (LLMs), Spike-driven Language Models (SLMs), Quantization, Memory Footprint, Embedded Systems, Design Automation.

## I. INTRODUCTION

Transformer-based networks [1] have achieved state-of-the-art performance (e.g., accuracy) in diverse machine learning (ML)-based applications, including solving diverse natural language tasks [2]–[7]. In recent years, transformer-based large language models (LLMs) have demonstrated significant improvements in extending the capabilities of natural language models [2]–[4], thereby making it possible to produce high-quality language-based understanding and responses to the given inputs. Therefore, their adoption in resource-constrained embedded devices is highly in demand and actively being pursued for enabling customized and personalized systems [8]. However, their large compute costs, huge memory footprints, and high processing power/energy make it challenging for their embedded deployments.

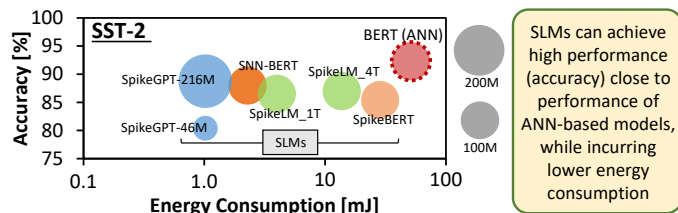


Fig. 1. Current trends of performance (i.e., accuracy), number of weight parameters (note, M denotes millions  $[10^6]$  of parameters), and energy consumption of SLMs [14]–[18] on the sentiment analysis task with the SST-2 dataset [19].

In the other domain, the advancements of spiking neural networks (SNNs) have demonstrated promising power/energy-efficient alternative to artificial neural network (ANN) algorithms, because of their sparse spike-driven operations [9] [10]. Therefore, recent works leveraged spike-driven operations for LLMs to reduce the processing power/energy requirements, i.e., so-called *Spike-driven Language Models (SLMs)*; see Fig. 1. However, their memory footprints are still too large for embedded deployments. To reduce memory footprints of spike-driven models, quantization is one of the prominent methods [11]–[13], because it can effectively reduce memory footprints with slightly yet acceptable accuracy degradation. However, manually determining an appropriate quantization setting for any given SLM requires huge design time and large power/energy consumption. Therefore, this approach is laborious and *not scalable* for compressing different SLMs for different possible performance and memory constraints. Moreover, existing ANN quantization frameworks cannot be employed directly for SLMs due to the fundamental differences in synaptic and neuronal operations between ANNs and SNNs.

Such conditions lead us to **the research problem** targeted in this paper, i.e., *how can we efficiently quantize any given pre-trained SLM, while maintaining high performance (e.g., accuracy) and meeting the memory constraint?* A solution to this problem may advance the design automation for efficient embedded implementation of SLMs.

### A. State-of-the-art of SLMs and Their Limitations

SLM development is a relatively new research avenue, hence the state-of-the-art works still focus on achieving high performance (e.g., accuracy), such as SpikeBERT [20], SpikingBERT [15], SNN-BERT [16], SpikeLM [14], SpikeLLM [17], and SpikeGPT [18]. Specifically, spike-driven BERTs [15] [16] [20] leverage BERT networks from ANN domain and apply spiking neuronal dynamics on them, while employing different techniques, such as knowledge distillation [15] and input coding enhancements [16]. SpikeLM [14] and SpikeLLM [17] target to

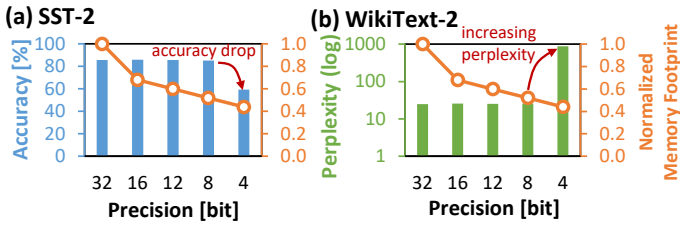


Fig. 2. Performance profiles of the pre-trained SpikeGPT-216M after uniformly quantizing its weight parameters in its attention blocks across different precision levels for different tasks: (a) sentiment classification on the SST-2 dataset [19], and (b) perplexity on the WikiText-2 dataset [21]. Note, a lower perplexity score represents a better text generation performance.

scale up spiking neuronal dynamics to large models (e.g., up to 70 billions of weight parameters for SpikeLLM). Meanwhile, SpikeGPT [18] targets at reducing the computational complexity in SLMs by replacing the spike-driven transformer modules with the spike-driven receptance weighted key value (SRWKV) modules, while maintaining the high performance. *These state-of-the-art highlight that the efforts for quantizing SLMs have not been comprehensively explored.*

### B. A Case Study and Associated Research Challenges

To show the potentials and challenges in quantizing SLMs, we perform an experimental case study. Here, we apply uniform quantization on the weight parameters of the pre-trained SpikeGPT-216M [18] with the same precision level across its attention blocks, then employ the quantized model for solving the sentiment analysis on the SST-2 dataset [19] and evaluating the perplexity on the WikiText-2 dataset [21]. Further details of experiments are provided in Sec. IV, and the experimental results are presented in Fig. 2. These results show that, the post-training quantization (PTQ) scheme leads to significant memory reduction, while preserving high performance (e.g., accuracy and perplexity) when employed with appropriate quantization. Otherwise, it leads to notable performance degradation.

Furthermore, these observations also expose several research challenges, as follows.

- Quantization process should handle different network complexity levels (e.g., number of layers) efficiently.
- Quantization process should be able to meet different possible performance (e.g., accuracy) and memory constraints, thus making it practical for diverse applications.
- Quantization process should minimize manual intervention to increase its scalability for handling different networks, performance requirements, and memory budgets.

### C. Our Novel Contributions

To address the targeted problem and research challenges, we propose *QSLM*, a novel framework that performs automated Quantization for compressing pre-trained Spike-driven Language Model (SLM) to meet the performance (e.g., accuracy) and memory constraints. To achieve this, QSLM performs the following key steps; see an overview in Fig. 3.

- **Network Model Analysis (Sec. III-A):** It aims to identify the structure of the given pre-trained model, determine the network hierarchy to be considered for quantization search,

and investigate the sensitivity of each block of the network under quantization on the performance (e.g., accuracy).

- **Tiered Search Strategy for Quantization (Sec. III-B):** It aims to perform automated quantization and evaluation for the model candidates under different phases (e.g., global-, block-, and module-level quantization, subsequently) based on the network hierarchy and the sensitivity analysis, while considering the performance and memory constraints.
- **Quantization Setting Selection (Sec. III-C):** It selects the final quantization setting from the candidates by leveraging our trade-off function that quantifies the candidates' benefits based on their performance and memory footprint.

**Key Results:** We implement the QSLM framework using PyTorch and then run it on the Nvidia RTX A6000 multi-GPU machine. Experimental results show that QSLM provides effective quantization settings for SLMs. It saves by up to 86.5% of memory footprint, reduces by up to 20% of power consumption, and maintain high performance across different tasks (i.e., by up to 84.4% accuracy of sentiment classification on the SST-2 and 23.2 perplexity score of text generation on the WikiText-2) close to the non-quantized model, while meeting the performance and memory constraints. These results show the potential of QSLM framework for enabling embedded implementation of SLMs.

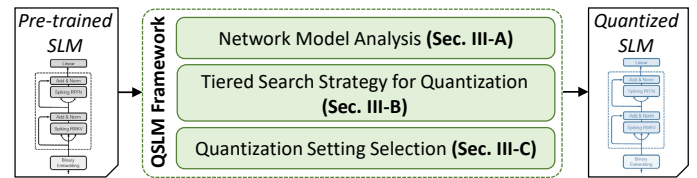


Fig. 3. Overview of our novel contributions.

## II. BACKGROUND

**SNNs:** An SNN model design typically encompasses spiking neurons, network architecture, neural/spike coding, and learning rule [11] [22]. Recent SNN developments in software [23]–[27] and hardware [28]–[35] have advanced the practicality of SNNs for diverse ultra-low power/energy application use-cases.

**SLMs:** Recently, several state-of-the-art SLMs have been proposed in the literature, such as SpikeBERT [20], SpikingBERT [15], SNN-BERT [16], SpikeLM [14], SpikeLLM [17], and SpikeGPT [18]. In this work, we consider SpikeGPT as the potential model candidate for embedded systems since it offers competitive performance with the lowest energy consumption due to its reduced computational complexity; see Fig. 1. Specifically, SpikeGPT replaces traditional self-attention mechanism with Spiking Receptance Weighted Key Value (SRWKV) and Spiking Receptance Feed-Forward Networks (SRFFN).

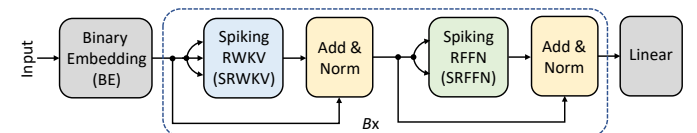


Fig. 4. Overview of the SpikeGPT architecture.  $B$  is the number of attention blocks. For instance, the pre-trained SpikeGPT-216M has  $B=18$  blocks [18].

SRWKV leverages element-wise products rather than matrix-matrix multiplication, hence reducing the computational cost

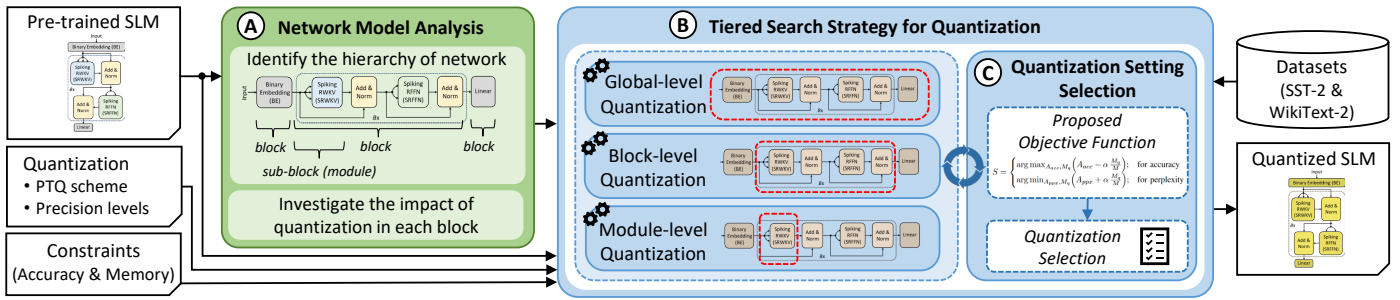


Fig. 5. Our QSLM framework showing its key steps: network model analysis, tiered search strategy for quantization, and quantization setting selection.

TABLE I

THE ARCHITECTURAL HIERARCHY OF THE SPIKEGPT-216M [18]. NOTE, THE ATTENTION PARAMETERS IN SRWKV INCLUDE  $K$ ,  $V$ , AND  $R$ , WHICH DENOTE KEY, VALUE, AND RECEPTANCE, RESPECTIVELY.

Block	Sub-Block (Module)	Number of Parameters	Quantity	Total Number of Parameters
Input	Embedding	38.6M	1	38.6M
	Layer Norm.	1.5K		
Attention	Layer Norm.	3K	18	138.2M
	SRWKV	2.4M		
	SRFFN	5.3M		
Output	Layer Norm.	1.5K	1	38.6M
	Head	38.6M		

of the attention mechanism. Meanwhile, SRFFN is employed to replace the conventional feed-forward network (FFN) with a spiking-compatible version. Its network architecture is illustrated in Fig. 4 and summarized in Table I. If the data have been processed through all network layers, the model either employs a classification head for natural language understanding (NLU) or a generation head for natural language generation (NLG).

#### A. SNN Quantization

There are two possible schemes for quantizing SNN models, namely *Quantization-aware Training (QAT)* and *Post-Training Quantization (PTQ)* [13] [36]. QAT quantizes an SNN model during the training phase based on the given precision level. Meanwhile, PTQ quantizes the pre-trained SNN model with the given precision level. In this work, we consider the PTQ scheme since it avoids the expensive training costs, such as the computational time, memory, and power/energy consumption [37]. To realize this, we employ the simulated quantization approach to enable fast design space exploration and provide representative results in performance (e.g., accuracy) and power/energy consumption saving [38].

### III. THE QSLM FRAMEWORK

We propose the novel QSLM framework to solve the targeted problem and related challenges, whose overview is presented in Fig. 5. It employs (A) *network model analysis* to identify the model structure and identify its block sensitivity under quantization, (B) *tiered search strategy* to systematically perform quantization on the model, and (C) *quantization setting selection* that considers performance and memory constraints. Details of its key steps are discussed in the following sub-sections.

#### A. Network Model Analysis

To perform effective quantization, it is important to apply appropriate precision levels on the weight parameters of the

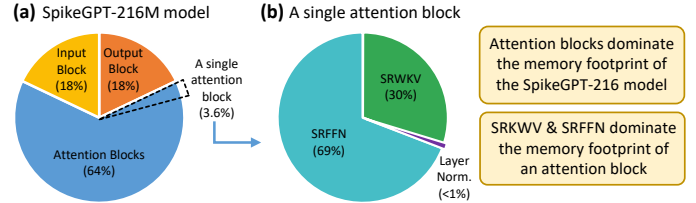


Fig. 6. Proportion of the memory footprint for (a) the SpikeGPT-216M model with its blocks, and (b) a single attention block with its sub-blocks/modules.

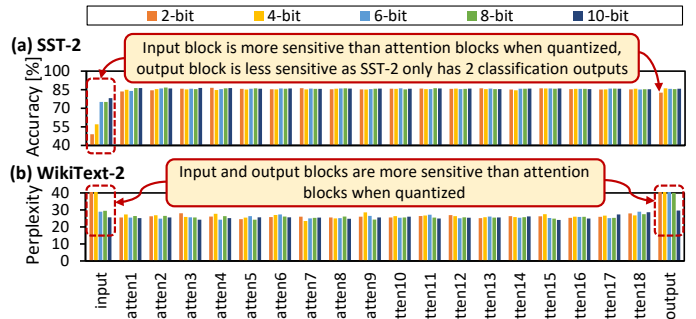


Fig. 7. Results of block-wise quantization in SpikeGPT-216M across different precision levels for (a) accuracy of sentiment classification task on the SST-2 dataset, and (b) perplexity of text generation task on the WikiText-2 dataset. Note, a lower perplexity score means a better performance of text generation.

model. Therefore, *this step targets to understand the network structure of the model, identify its architectural hierarchy for quantization search, and investigate its block sensitivity under quantization on the performance*, through the following ideas.

- We identify blocks in the network model that can be quantized. Typically, they are categorized as the *input*, *attention*, and *output blocks*.
- For each block, we identify the sub-blocks (modules) and the respective number of weights to estimate the memory saving potentials; see Table I and Fig. 6 for SpikeGPT-216M.
- Then, we investigate the block sensitivity under quantization by applying different precision levels to individual block and evaluating the performance (e.g., accuracy). It is useful for devising a suitable strategy for quantization search.

For instance, we conduct experiments that apply different weight precision levels on each block of the SpikeGPT-216M for sentiment analysis on the SST-2 and text generation on the WikiText-2. Experimental results are presented in Fig. 7, from which we make the following key observations.

- The input and output blocks are more sensitive than the attention blocks, since the loss of information from quantization in these blocks lead to notable performance degradation.

Therefore, *the input and output blocks should be carefully quantized to maximize memory reduction while ensuring high performance (e.g., accuracy).*

- The attention blocks are less sensitive than the input/output block. Considering that the attention blocks dominate the memory footprint, quantizing them potentially lead to significant memory reduction. Therefore, *quantizing the attention blocks is beneficial to achieve significant memory reduction.*

These observations are then leveraged in Sec. III-B to enable automated quantization process.

### B. Tiered Search Strategy for Quantization

This step aims to enable an automated quantization process to maximize the memory reduction, while meeting both performance constraint ( $const_A$ ) and memory constraint ( $const_M$ ). To obtain this, *we propose a tiered search strategy that applies a certain bit precision level ( $b$ ) to the targeted weights from the highest-level network hierarchy to the lowest one (e.g., global-level, block-level, and module-level quantization, subsequently).* Its key steps are described below (pseudocode in Alg. 1 and 2).

- **Global-level quantization:** We uniformly quantize all blocks in the model based on the pre-defined list of precision levels ( $b$ ), such as  $b \in \{16, 14, 12, \dots, 4\}$ . Here, we orderly apply  $b$  value from the largest to the smallest ones, while evaluating if the quantized model meets both  $const_A$  and  $const_M$ .
  - If both constraints are met, then the investigated precision level  $b$  is recorded as the quantization candidate ( $candQ$ ).
  - If both constraints are not met, then the selected precision is set back to the last acceptable precision (from index- $I_{last}$  of list  $b$ ). Then, we move to *block-level quantization*.
- **Block-level quantization:** We quantize each block in the model with lower precision than the previously applied one in the global-level step. Then, we subsequently apply lower precision based on the list  $b$ , while performing evaluation.
  - If both constraints are met, then the investigated precision level  $b$  is recorded as the setting for the respective block, and used to update the candidate  $candQ$ .
  - If both constraints are not met, then the selected precision for the respective block is set back to the last acceptable precision level (from index- $I_{last2}$  of list  $b$ ). Afterward, we move to *module-level quantization*.
- **Module-level quantization:** We quantize each module in the attention blocks with lower precision than the previously applied one in the block-level step. We further apply lower precision based on the list  $b$ , while performing evaluation.
  - If both constraints are met, then the investigated precision level  $b$  is recorded as the quantization setting for the respective module, and used to update the candidate  $candQ$ .
  - If both constraints are not met, then the precision level  $b$  for the respective module is set back to the last acceptable precision level (from index- $I_{last3}$  of list  $b$ ).

### C. Quantization Setting Selection

The tiered search strategy may obtain multiple quantization candidates that meet  $const_A$  and  $const_M$ . To select the most appropriate solution, we quantify the benefit of the candidates

---

### Algorithm 1 Tiered search strategy for quantization

---

**INPUT:** (1) Pre-trained model ( $Net$ ), its performance ( $P$ ) and memory footprint ( $M$ ); (2) Pre-defined bit precision levels  $b: b \in \{16, 14, 12, \dots, 4\}$ , and its number of precision levels ( $N_b$ ); (3) Number of blocks in the model ( $N_k$ ); (4) Number of modules in the attention block ( $N_m$ ); (5) Constraints: performance constraint ( $const_A$ ), and memory constraint ( $const_M$ );

**OUTPUT:** Quantized model ( $Net_q$ );

**BEGIN**

**Initialization:**

```

1:  $c = 0$ ;
2:  $candQ[c, :, :] = 32$ ;
3:  $P, M = \text{test}(Net, candQ[c, :, :])$ ;
4:  $cStat[c].perf = P$ ;
5:  $cStat[c].mem = M$ ;

```

**Process:**

```

// Global-level quantization
6: for ( $i = 0; i < N_b; i++$ ) do
7:    $c = c + 1$ ;
8:    $candQ[c, :, :] = b[i]$ ;
9:    $Net_t = \text{quantize}(Net, candQ[c, :, :])$ ;
10:   $cStat[c], X = \text{eval}(Net_t, P, M, const_A, const_M)$ ; // Alg. 2
11:  if ( $X == \text{'constraints are met'}$ ) then
12:     $cStat[c].met = \text{'true'}$ ;
13:     $I_{last} = i$ ;
14:  else
15:     $cStat[c].met = \text{'false'}$ ;
16:     $I_{tmp} = I_{last}$ ;

// Block-level quantization
17: for ( $k = 0; k < N_k; k++$ ) do
18:  for ( $i = I_{tmp}; i < N_b; i++$ ) do
19:     $c = c + 1$ ;
20:     $candQ[c, k, :] = b[i]$ ;
21:     $Net_t = \text{quantize}(Net, candQ[c, :, :])$ ;
22:     $cStat[c], X = \text{eval}(Net_t, P, M, const_A, const_M)$ ; // Alg. 2
23:    if ( $X == \text{'constraints are met'}$ ) then
24:       $cStat[c].met = \text{'true'}$ ;
25:       $I_{last2}[k] = i$ ;
26:    else
27:       $cStat[c].met = \text{'false'}$ ;
28:       $I_{tmp2}[k] = I_{last2}[k]$ ;

// Module-level quantization
29: for ( $k = 1; k < (N_k - 1); k++$ ) do
30:  for ( $m = 0; m < N_m; m++$ ) do
31:    for ( $i = I_{tmp2}[k]; i < N_b; i++$ ) do
32:       $c = c + 1$ ;
33:       $candQ[c, k, m] = b[i]$ ;
34:       $Net_t = \text{quantize}(Net, candQ[c, :, :])$ ;
35:       $cStat[c], X = \text{eval}(Net_t, P, M, const_A, const_M)$ ; // Alg. 2
36:       $cStat[c].score = S_{tmp}$ ;
37:      if ( $X == \text{'constraints are met'}$ ) then
38:         $cStat[c].met = \text{'true'}$ ;
39:         $I_{last3}[k, m] = i$ ;
40:      else
41:         $cStat[c].met = \text{'false'}$ ;
42:   $cand_{fin} = \text{select}(candQ, \max(cStat[:, :].score), cStat[:, :].met)$ ;
43:   $Net_q = \text{quantize}(Net, cand_{fin})$ ;
44:  return  $Net_q$ ;

```

**END**

---

considering their performance (e.g., accuracy) and memory saving, and then select the one with the highest score ( $S$ ). To do this, *we propose a performance-and-memory trade-off function, that can be expressed as Eq. 1.* Here,  $A_{acc}$  denotes accuracy for classification task and  $A_{ppx}$  denotes perplexity for generation task;  $M$  and  $M_q$  denote memory footprints for the original non-quantized model and quantized model, respectively; and  $\alpha$  denotes the user-defined adjustment factor. In the classification task, it aims to maximize the score  $S$ , that is proportional to the accuracy, since higher accuracy is better. In the generation

**Algorithm 2** Evaluation of the quantized model candidate

**INPUT:** (1) Performance ( $P$ ) and memory footprint ( $M$ ) of the original non-quantized model; (2) Input model ( $Net_{tmp}$ ); (3) Constraints: performance (i.e., accuracy/perplexity) constraint ( $const_A$ ), and memory constraint ( $const_M$ ); (4) Candidate index ( $c$ );

**OUTPUT:** (1) Characteristics of the model candidates ( $cStat$ ); (2) Status if constraints are met ( $X$ : 'true'/'false');

**BEGIN**

**Process:**

- 1:  $P_{tmp}, M_{tmp} = \text{test}(Net_{tmp})$ ;
- 2:  $S_{tmp} = \text{calc\_score}(P_{tmp}, M_{tmp})$ ; // Eq. 1
- 3:  $X = \text{check}(P, M, const_P, const_M, P_{tmp}, M_{tmp})$ ;
- 4:  $cStat[c].perf = P_{tmp}$ ;
- 5:  $cStat[c].mem = M_{tmp}$ ;
- 6:  $cStat[c].score = S_{tmp}$ ;
- 7: **return**  $cStat, X$ ;

**END**

task, it aims to minimize the score  $S$ , since lower perplexity is better. A candidate with larger memory than other candidates will penalize more the score  $S$ . Furthermore, perplexity score  $A_{ppx}$  can be calculated using Eq. 2, with  $N_T$  is the number of words (tokens) in the sequence, and  $P(w_i | w_{<i})$  is the model's predicted probability of word  $w_i$  given the previous words.

$$S = \begin{cases} \arg \max_{A_{acc}, M_q} \left( A_{acc} - \alpha \frac{M_q}{M} \right); & \text{for accuracy} \\ \arg \min_{A_{ppx}, M_q} \left( A_{ppx} + \alpha \frac{M_q}{M} \right); & \text{for perplexity} \end{cases} \quad (1)$$

$$A_{ppx} = \exp \left( -\frac{1}{N_T} \sum_{i=1}^{N_T} \log P(w_i | w_{<i}) \right) \quad (2)$$

#### IV. EVALUATION METHODOLOGY

To evaluate the QSLM framework, we develop its PyTorch-based implementation, then run it on the Nvidia RTX A6000 multi-GPU machine; see Fig. 8. For the baseline non-quantized model we consider the state-of-the-art pre-trained SpikeGPT-216M [18] that has been trained with 5B tokens from the OpenWebText dataset [39]. We use its publicly available pre-trained model and codes from the original authors, and then reproduce the fine-tuning and testing phases with their default hyperparameter settings on targeted tasks. In the evaluation, we consider the following tasks: (1) a sentiment classification task on the SST-2 dataset [19], and (2) a text generation task on the WikiText-2 dataset [21]. Under the baseline settings, we achieve accuracy of 85.7% for the sentiment classification task, and perplexity score of 26.5 for the text generation task. Here, we consider different sets of constraints to investigate the performance of QSLM under different constraint cases.

- In sentiment classification task, case-a1:  $const_A = 2\%$  and  $const_M = 400\text{MB}$ ; case-a2:  $const_A = 5\%$  and  $const_M = 400\text{MB}$ ; and case-a3:  $const_A = 5\%$  and  $const_M = 420\text{MB}$ .
- In text generation task, case-b1:  $const_A = 1$  and  $const_M = 400\text{MB}$ ; case-b2:  $const_A = 4$  and  $const_M = 400\text{MB}$ ; and case-b3:  $const_A = 4$  and  $const_M = 420\text{MB}$ .

Note,  $const_A$  denotes the maximum acceptable accuracy degradation or perplexity increase, while  $const_M$  denotes the maximum acceptable memory footprint. Furthermore, we also perform ablation study for investigating the impact of different  $\alpha$  values with  $\alpha \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$ . The experiments

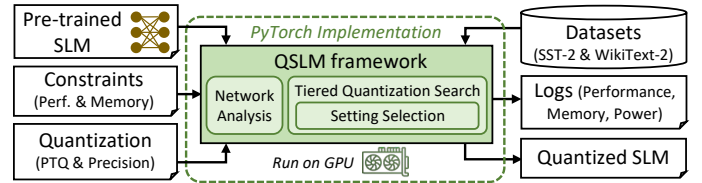


Fig. 8. Experimental setup for the evaluation

evaluate several metrics, such as accuracy for sentiment classification task, perplexity score for text generation task, memory footprint, and power consumption (using *nvidia-smi* utility).

## V. RESULTS AND DISCUSSION

### A. Reducing Memory while Maintaining High Performance

Experimental results for sentiment classification task are provided in Fig. 9(a). These results show that, QSLM effectively reduces memory footprint of the baseline model across different scenarios (i.e., different sets of constraints and different  $\alpha$  values), while meeting both accuracy and memory constraints. Our key observations are the following.

- For case-a1 in Fig. 9(a.1): QSLM achieves 84.4% accuracy and reduces 85.7% memory footprint; see ①.
- For case-a2 in Fig. 9(a.2): QSLM achieves 81.3% accuracy and reduces 86.5% memory footprint; see ②.
- For case-a3 in Fig. 9(a.3): QSLM achieves 81.3% accuracy and reduces 86.5% memory footprint; see ③.

Meanwhile, experimental results for text generation task are provided in Fig. 9(b). These results also show that, QSLM effectively reduces memory footprint of the baseline model across different scenarios (i.e., different sets of constraints and different  $\alpha$  values), while meeting both perplexity and memory constraints. Our key observations are the following.

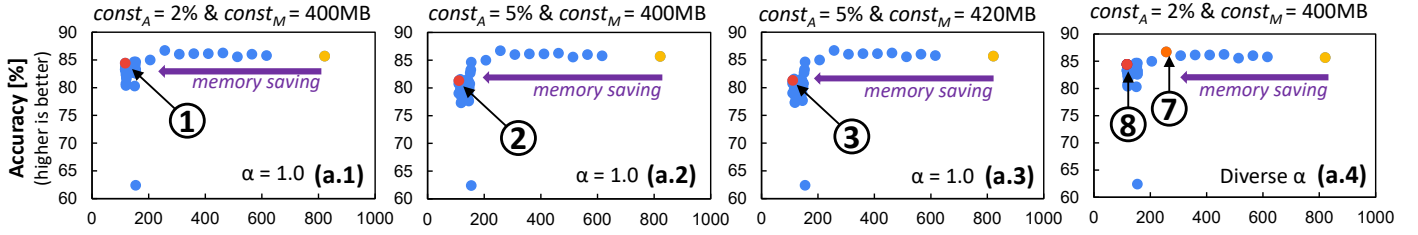
- For case-b1 in Fig. 9(b.1): QSLM achieves 24.6 perplexity score and reduces 68.7% memory footprint; see ④.
- For case-b2 in Fig. 9(b.2): QSLM achieves 23.2 perplexity score and reduces 62.4% memory footprint; see ⑤.
- For case-b3 in Fig. 9(b.3): QSLM achieves 24.4 perplexity score and reduces 68.7% memory footprint; see ⑥.

These significant memory savings while preserving high accuracy or low perplexity can be obtained due to the systematic quantization approach in our QSLM framework. Specifically, QSLM leverages the block sensitivity information from model analysis to guide the quantization search, then performs tiered search strategy to carefully apply different precision levels on different network blocks/modules, while ensuring the selected model candidates always meet the given constraints (i.e.,  $const_A$  and  $const_M$ ) by leveraging a performance-and-memory trade-off function and the given constraints.

### B. Reduction of Power Consumption

Experimental results for power consumption of the baseline model and the QSLM model candidates that meet both performance and memory constraints are provided in Fig. 10. For the sentiment classification task, QSLM model candidates can reduce the power consumption by 2.6%-20%. Meanwhile, for the text generation task, QSLM model candidates can reduce

### (a) Sentiment Classification Task on the SST-2



### (b) Text Generation Task on the WikiText-2

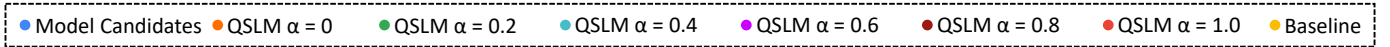
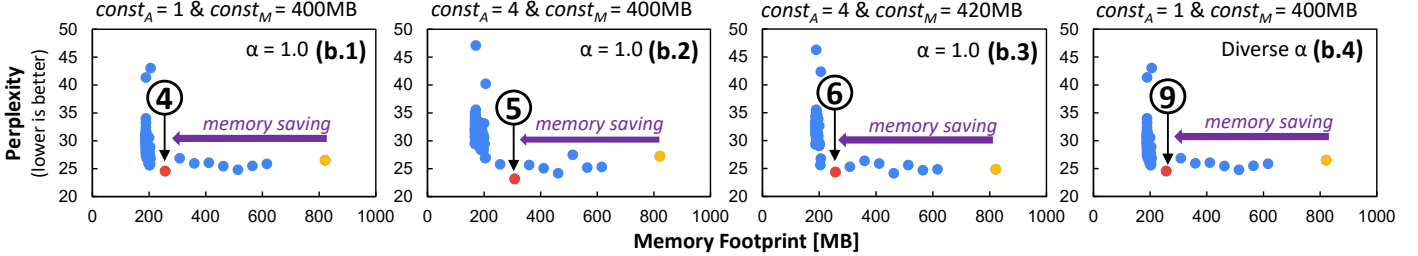


Fig. 9. Experimental results of (a) sentiment classification task on the SST-2 for different sets of constraints (a1-a3) and diverse  $\alpha$  (a4); and (b) text generation task on the WikiText-2 for different sets of constraints (b1-b3) and diverse  $\alpha$  (b4).

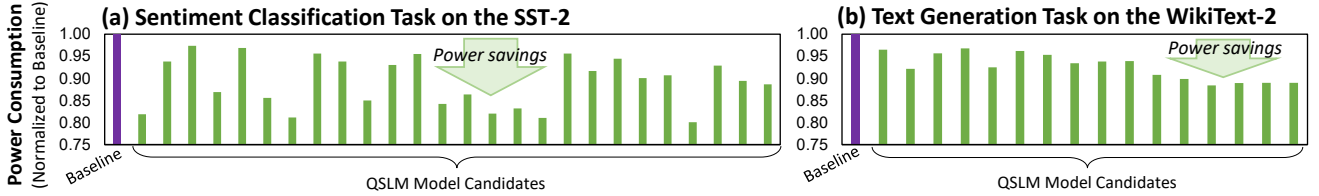


Fig. 10. Experimental results of power consumption incurred by the baseline model and our QSLM model candidates that meet both constraints for (a) sentiment classification task on the SST-2, and (b) text generation task on the WikiText-2.

the power consumption by 3.2%-11.6%. These power savings come from the reduction of precision levels in the weight parameters of the quantized models, thereby incurring lower computational and memory power to complete the processing, as compared to the baseline non-quantized model. Furthermore, these results also demonstrate that, QSLM effectively optimizes power consumption, while meeting both performance and memory constraints (i.e.,  $const_A$  and  $const_M$ ).

#### C. Impact of Different $\alpha$ Values on the Model Selection

Experimental results for investigating the impact of different  $\alpha$  values on the model selection are provided in Fig. 9(a.4) for sentiment classification task and Fig. 9(b.4) for text generation task. These results show that, different  $\alpha$  values may lead to different model selection, as summarized below.

- In the sentiment classification task,  $\alpha = 0$  guides the QSLM search strategy to put the memory aspect as non-priority, and hence leading the selection process toward a model with higher accuracy and higher memory footprint, as pointed by ⑦ in Fig. 9(a.4). Meanwhile, the other investigated  $\alpha$  values guide the QSLM search strategy to adjust the priority level of memory aspect proportional to the respective  $\alpha$  value. In this case study, QSLM search strategy selects a quantized model candidate that is pointed by ⑧ in Fig. 9(a.4). These results demonstrate that, our performance-and-memory trade-off function in QSLM effectively helps selection of quantized

model based on the priority of memory footprint relative to performance (e.g., accuracy).

- In the text generation task, all investigated  $\alpha$  values lead the QSLM search strategy to select a model with 24 perplexity score and 68.7% memory saving, as shown by ⑨ in Fig. 9(b.4). These results demonstrate that, there are some conditions that QSLM search strategy finds a relatively dominant quantized model in performance (e.g., perplexity), hence adjusting  $\alpha$  with small values does not change the final selection for the quantized model.

## VI. CONCLUSION

In this paper, we propose the novel QSLM framework for performing automated quantization on the pre-trained SLMs. Our QSLM significantly reduces memory footprint by up to 86.5%, decreases power consumption by up to 20%, preserves high performance across different tasks (i.e., by up to 84.4% accuracy for the SST-2 dataset and 23.2 perplexity score for the WikiText-2 dataset), while meeting the given accuracy and memory constraints. These results also demonstrate that our QSLM successfully advances the efforts in enabling efficient design automation for embedded implementation of SLMs.

## ACKNOWLEDGMENT

This work was partially supported by the NYUAD Center for CyberSecurity (CCS), funded by Tamkeen under the NYUAD Research Institute Award G1104.

## REFERENCES

- [1] 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 (NIPS)*, vol. 30, no. 1, pp. 261–272, 2017.
- [2] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.
- [3] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," *arXiv preprint arXiv:2402.06196*, 2024.
- [4] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, Mar. 2024.
- [5] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu *et al.*, "A survey on vision transformer," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 45, no. 1, pp. 87–110, 2022.
- [6] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.
- [7] R. V. W. Putra, S. Iftikhar, and M. Shafique, "Qsvit: A methodology for quantizing spiking vision transformers," in *2025 International Joint Conference on Neural Networks (IJCNN)*, 2025, pp. 1–8.
- [8] A. El Mir, L. T. Luoga, B. Chen, M. A. Hanif, and M. Shafique, "Democratizing mlms in healthcare: Tinyllava-med for efficient healthcare diagnostics in resource-constrained settings," in *2024 IEEE International Conference on Image Processing Challenges and Workshops (ICIPCW)*. IEEE, 2024, pp. 4164–4170.
- [9] C. Bartolozzi, G. Indiveri, and E. Donati, "Embodied neuromorphic intelligence," *Nature communications*, vol. 13, no. 1, p. 1024, 2022.
- [10] R. V. W. Putra, P. Wickramasinghe, and M. Shafique, "Enabling efficient processing of spiking neural networks with on-chip learning on commodity neuromorphic processors for edge ai systems," in *2025 International Joint Conference on Neural Networks (IJCNN)*, 2025, pp. 1–8.
- [11] R. V. W. Putra and M. Shafique, "Fspinn: An optimization framework for memory-efficient and energy-efficient spiking neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 11, pp. 3601–3613, 2020.
- [12] N. Rathi, P. Panda, and K. Roy, "Stdp-based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 38, no. 4, pp. 668–677, April 2019.
- [13] R. V. W. Putra and M. Shafique, "Q-spinn: A framework for quantizing spiking neural networks," in *International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–8.
- [14] X. Xing, Z. Zhang, Z. Ni, S. Xiao, Y. Ju, S. Fan, Y. Wang, J. Zhang, and G. Li, "Spikelm: Towards general spike-driven language modeling via elastic bi-spiking mechanisms," in *International Conference on Machine Learning (ICML)*. PMLR, 2024, pp. 54 698–54 714.
- [15] M. Bal and A. Sengupta, "Spikingbert: Distilling bert to train spiking language models using implicit differentiation," in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 38, no. 10, 2024, pp. 10998–11006.
- [16] Q. Su, S. Mei, X. Xing, M. Yao, J. Zhang, B. Xu, and G. Li, "Snnbert: Training-efficient spiking neural networks for energy-efficient bert," *Neural Networks*, vol. 180, p. 106630, 2024.
- [17] X. Xing, B. Gao, Z. Liu, D. A. Clifton, S. Xiao, W. Zhang, L. Du, Z. Zhang, G. Li, and J. Zhang, "Spikellm: Scaling up spiking neural network to large language models via saliency-based spiking," in *The 13th International Conference on Learning Representations (ICLR)*, 2024.
- [18] R.-J. Zhu, Q. Zhao, G. Li, and J. Eshraghian, "SpikeGPT: Generative pre-trained language model with spiking neural networks," *Transactions on Machine Learning Research (TMLR)*, 2024.
- [19] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *International Conference on Learning Representations (ICLR)*, 2019.
- [20] C. Lv, T. Li, J. Xu, C. Gu, Z. Ling, C. Zhang, X. Zheng, and X. Huang, "Spikbert: A language spikformer learned from bert with knowledge distillation," *arXiv preprint arXiv:2308.15122*, 2024.
- [21] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *International Conference on Learning Representations (ICLR)*, 2017.
- [22] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, and T. Masquelier, "Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron," *Frontiers in Neuroscience*, vol. 13, p. 625, 2019.
- [23] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.
- [24] N. Rathi, I. Chakraborty, A. Kosta, A. Sengupta, A. Ankit, P. Panda, and K. Roy, "Exploring neuromorphic computing based on spiking neural networks: Algorithms to hardware," *ACM CSUR*, vol. 55, no. 12, 2023.
- [25] R. V. W. Putra and M. Shafique, "Topspark: a timestep optimization methodology for energy-efficient spiking neural networks on autonomous mobile agents," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 3561–3567.
- [26] S. S. Chowdhury, N. Rathi, and K. Roy, "Towards ultra low latency spiking neural networks for vision and sequential tasks using temporal pruning," in *European Conference on Computer Vision (ECCV)*. Springer, 2022, pp. 709–726.
- [27] R. V. W. Putra and M. Shafique, "Spikenas: A fast memory-aware neural network search framework for spiking neural network-based embedded ai systems," *IEEE Transactions on Artificial Intelligence (TAI)*, pp. 1–12, 2025.
- [28] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, no. 10, pp. 1537–1557, Oct 2015.
- [29] A. Roy, S. Venkataramani, N. Gala, S. Sen, K. Veezhinathan, and A. Raghunathan, "A programmable event-driven architecture for evaluating spiking neural networks," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, July 2017, pp. 1–6.
- [30] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan 2018.
- [31] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen, "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2019.
- [32] C. Frenkel, M. Lefebvre, J. Legat, and D. Bol, "A 0.086-mm<sup>2</sup> 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos," *IEEE Transactions on Biomedical Circuits and Systems (TBCAS)*, vol. 13, no. 1, pp. 145–158, Feb 2019.
- [33] C. Frenkel, J.-D. Legat, and D. Bol, "Morphic: A 65-nm 738k-synapse/mm<sup>2</sup> quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning," *IEEE Trans. on Biomedical Circuits and Systems (TBCAS)*, vol. 13, no. 5, pp. 999–1010, 2019.
- [34] SynSense. Dynap-cnn: The world's first fully scalable, event-driven neuromorphic processor with up to 1m configurable spiking neurons and direct interface with external dvs. [Online]. Available: <https://www.synsense.ai/products/dynap-cnn/>
- [35] BrainChip. Akida neural processor soc. [Online]. Available: <https://brainchip.com/akida-neural-processor-soc/>
- [36] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv*, vol. 1806.08342, 2018.
- [37] Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort, "Spinquant: LLM quantization with learned rotations," in *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.
- [38] M. van Baalen, B. Kahne, E. Mahurin, A. Kuzmin, A. Skliar, M. Nagel, and T. Blankevoort, "Simulated quantization, real power savings," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 2757–2761.
- [39] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, "The Pile: An 800gb dataset of diverse text for language modeling," *arXiv preprint arXiv:2101.00027*, 2020.