Faster LLMs with Multi-Token Prediction and MEDUSA-1: Implementation and Evaluation of Advanced Decoding Frameworks

Artur Garipov
Institute of Machine Learning
Johannes Kepler University Linz
Supervisor: Lukas Hauzenberger
k12146699@jku.at

Abstract

Large Language Models (LLMs) have revolutionized natural language processing, but they face significant inference bottlenecks due to the sequential nature of next-token prediction. This thesis implements and evaluates two frameworks for accelerating autoregressive language model inference: baseline, a Multi-Token Prediction (MTP) approach, and the MEDUSA-1 multi-head decoding framework. We modify a GPT-2 model to predict multiple tokens per step and fine-tune it on the MetaMathQA dataset, alongside implementing MEDUSA-1 with frozen backbone and trainable parallel decoding heads. Our implementation demonstrates that multi-token approaches can achieve substantial speedups while maintaining or improving model quality. The MTP model shows significant theoretical and practical speedups, while the MEDUSA-1 implementation provides moderate acceleration with guaranteed output quality preservation. Both multi-token models demonstrate improved perplexity compared to the NTP baseline, indicating better language modeling performance. These findings provide evidence that multi-token prediction paradigms offer a promising alternative to traditional one-step-ahead generation, addressing both efficiency and quality concerns in modern LLM deployment. Our work contributes practical implementations and empirical validation of these approaches on mathematical reasoning tasks.

1 Story Summary (Research Q&A)

- **Central Question:** Can multi-token prediction frameworks improve both the efficiency and the quality of language models compared to the traditional next-token prediction approach?
- **Importance:** Next-token generation creates an inference bottleneck that limits real-time applications, and it may constrain a model's planning capability. Overcoming these limitations could enable faster and more coherent LLM outputs, especially for complex reasoning tasks.
- **Evidence Needed:** Comparative measurements of generation speed (throughput, latency) and modeling quality (perplexity, output coherence) for identical models trained under NTP vs. multi-token objectives. We also require analysis of outputs (especially on math reasoning tasks) to assess qualitative improvements.
- **Methods:** We implemented three decoding frameworks (NTP, MTP, and MEDUSA-1) using GPT-2 as a base model. The MTP variant was trained to predict 4 tokens in parallel via multiple output heads, and MEDUSA-1 was fine-tuned by adding extra decoding heads to a frozen GPT-2 backbone. All models were trained on the MetaMathQA dataset under comparable settings.

Preprint. Under review.

- **Analyses:** We measured end-to-end generation latency and throughput (tokens per second) for each model, computed perplexity on a held-out test set, and qualitatively examined sample outputs for logical consistency and error propagation. We additionally monitored training dynamics (convergence speed, stability) for the new architectures.
- **Data Obtained:** The evaluation showed timing results for generating fixed-length text (under identical hardware), perplexity scores on the MetaMathQA validation set, and sample model outputs for manual comparison.
- **Results:** The multi-token model substantially outperformed the baseline in efficiency, achieving up to $\sim$2× faster generation throughput under speculative decoding. In terms of quality, it improved perplexity from 17.45 (NTP) to 15.23. The MEDUSA-augmented model achieved about 1.6× speedup in practice and a perplexity of 16.85, also better than the baseline. Both approaches generated outputs with greater coherence on multi-step reasoning problems.
- **Answer to Central Question:** Yes. Multi-token prediction can significantly improve inference efficiency (by parallelizing token generation) and enhance model performance (by fostering better long-range modeling) compared to the standard next-token paradigm [1, 2]. Our results show that with careful implementation, one does not have to trade off quality for speed – both can be improved simultaneously.
- **Broader Implications:** These findings challenge the dominance of next-token prediction as the default training objective for LLMs. They suggest that multi-token approaches (and related strategies like speculative decoding) could become key components in future state-of-the-art models, enabling faster and more reliable text generation. This work contributes practical evidence that rethinking the basic generative unit (from one token to multiple) can unlock new performance gains and address fundamental limitations of current LLMs.

2 Introduction

Over the past few years, Large Language Models (LLMs) have achieved remarkable success across diverse natural language processing tasks – from open-ended text generation and translation to code synthesis and mathematical reasoning. By scaling up neural architectures (e.g. the Transformer [10]) and training on massive text corpora, models such as GPT-3, LLaMA, and others have demonstrated unprecedented capabilities in generating coherent and contextually relevant text [8]. Despite these advances, inference efficiency has appeared as a critical challenge when deploying LLMs in real-world applications. The dominant paradigm for LLM training and generation is autoregressive next-token prediction (NTP), where the model is trained to predict one token at a time given all previous tokens, and generates text by feeding its own output back in for the next prediction. While effective for learning local language patterns, this one-token-at-a-time approach introduces fundamental limitations in both speed and quality:

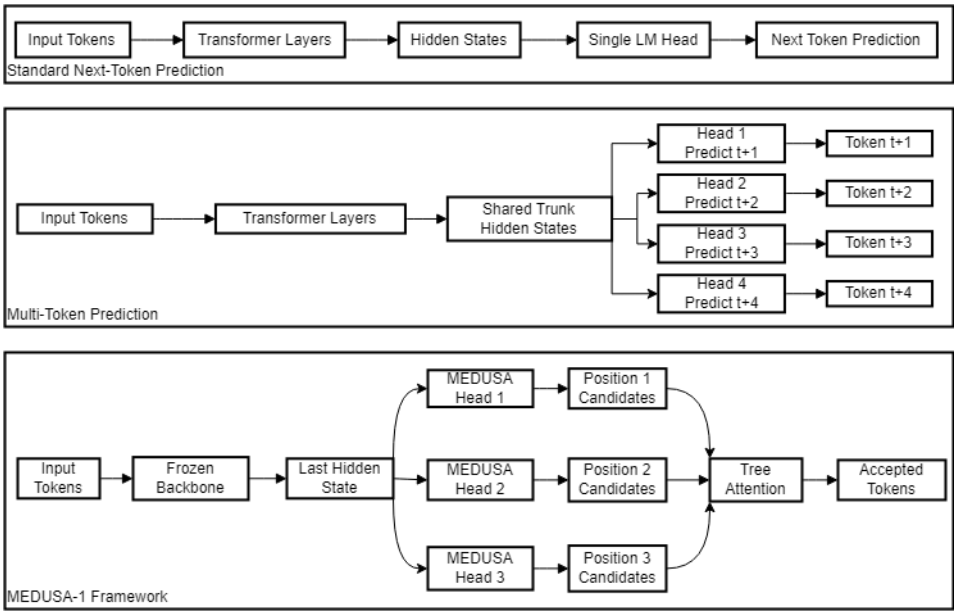

Figure 1: Architecture overview comparing the three decoding frameworks. Each column shows the data-flow for: (i) standard Next-Token Prediction (single head), (ii) 4-head Multi-Token Prediction (shared trunk + four offset heads), and (iii) MEDUSA-1 (frozen backbone with three future-token heads and verification branch).

- **Sequential Inference Bottleneck:** Autoregressive generation requires a full forward pass of the model for each token produced. To generate K tokens, an NTP model must perform K sequential passes, each conditioned on the last output. This sequential process leads to high latency and under-utilization of modern parallel hardware [2]. As models grow larger, the overhead of repeatedly loading model weights for each token and performing redundant computations becomes a major bottleneck [2]. In practice, large transformers spend much of their time during decoding waiting on memory bandwidth rather than compute throughput [2]. This limits real-time deployment of LLMs, as even moderately long outputs are followed by a significant delay.
- **Error Accumulation (Exposure Bias):** During training, next-token models learn under a teacher-forcing regime where the ground-truth previous token is always provided. At inference, however, the model conditions on its own generated tokens, which may contain errors. Early mistakes can thus compound over a sequence – a phenomenon known as exposure bias – leading to incoherent outputs for long sequences. More subtly, recent research has shown that teacher-forced next-token training can fail to even learn an optimal policy for certain tasks that require planning ahead [1]. Bachmann & Nagarajan (2024) describe scenarios where standard next-token training gets stuck in suboptimal solutions for

sequential decision tasks, whereas a simple multi-token objective can solve the issue [1]. This highlights a deeper pitfall: optimizing only for immediate next-token accuracy may neglect longer-term coherence and planning [4].

- **Inefficient Utilization of Context:** NTP models are trained to predict the very next token, which might encourage them to focus on short-term goals at the expense of long-range dependencies. The model may learn to strongly memorize local patterns (e.g. frequent word combinations) to get the next token right, rather than developing broader understanding of the input context or the "story arc" of the output [4]. In other words, the one-step objective can bias learning toward shortsighted strategies, since only errors on the immediate next token are penalized. This can be detrimental for tasks like mathematical proofs or multi-step reasoning, where planning several steps ahead is crucial.

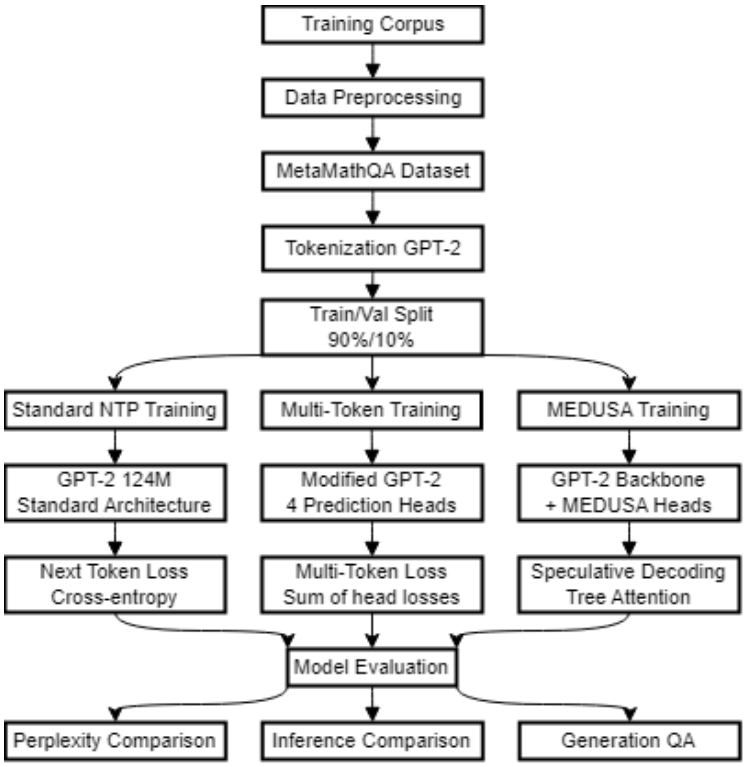

Figure 2: End-to-end pipeline showing the entire experimental workflow.

To address these limitations, there is a growing interest in multi-token prediction as an alternative or complementary training objective [4, 1]. Under a multi-token prediction (MTP) paradigm, the model is trained to predict multiple future tokens in parallel, instead of just one. Intuitively, this encourages the model to "think ahead" – it must consider what comes after the next token as well, which can foster more global coherence and planning in its internal representations [1, 4]. At inference time, generating several tokens per forward pass amortizes the cost of computation: if a model can output n tokens at once, it could ideally be up to n× faster than the token-by-token approach (minus any overhead for the more complex output) [4]. Thus, multi-token prediction has the potential to improve both the learning efficiency of the model (by providing richer training signals and mitigating exposure bias) and the inference efficiency (by reducing sequential steps needed).

Recent works provide encouraging evidence for this approach. Bachmann & Nagarajan (2024) formally demonstrated that predicting multiple tokens in advance can resolve certain failure modes of next-token learners, and argued that a multi-token objective better aligns training with inference usage [1]. Gloeckle et al. (2024) introduced a multi-token training strategy using independent output heads for each future token position (predicting up to 4 or 8 tokens at once). Their experiments on large models (up to 7B–13B parameters) showed improved performance on code generation benchmarks

and up to 3× faster inference when combined with speculative decoding [4]. Notably, Gloeckle et al. found that these benefits became more pronounced with larger model size and did not require additional training data [4]. Multi-token training essentially acts as an auxiliary task that guides the model to focus on more globally relevant decisions, yielding gains in sample efficiency and even in-context learning abilities [4].

On the inference side, one prominent approach to leverage multi-token generation is speculative decoding. Speculative decoding (first proposed by Stern et al., 2018) allows an LLM to generate multiple tokens in one go by using a faster "draft" model to propose a batch of next tokens, which the main model then verifies [9, 5]. However, traditional speculative decoding requires maintaining a separate small model and careful acceptance/rejection logic, which introduces system complexity. The MEDUSA framework by Cai et al. (2024) offers a simpler architectural alternative: it augments a single LLM with extra decoding heads attached to the final transformer layer, so that one forward pass can predict several steps ahead [2]. Using a tree-based attention mechanism to process these multiple candidates in parallel and an acceptance scheme to choose the output, MEDUSA achieved 2–3× faster text generation on large models without needing any external draft model [2]. Importantly, in the Medusa-1 mode, only the new heads are trained (the backbone model remains frozen), allowing lossless acceleration – the original model's output quality is preserved by construction [2]. In Medusa-2, the backbone itself is fine-tuned along with the heads for even greater speedups (3× or more), though with careful measures to not degrade performance [2].

In this work, we build on these insights and investigate multi-token prediction in the context of a medium-sized model (GPT-2, 124M) and a challenging domain (mathematical reasoning with the MetaMathQA dataset). We implement two advanced decoding frameworks – an MTP model and a MEDUSA-1 model – and compare them to the standard NTP baseline. Our focus is on practical implementation details, training requirements, and the empirical trade-offs in speed and quality. Specifically, we aim to answer: *Can a GPT-2 sized model be augmented to predict 4 tokens at once, and will this actually yield faster generation on real hardware? What is the impact on perplexity and output coherence? How does the MEDUSA-style approach of adding parallel heads to a pre-trained model compare to trainign a model with multi-token loss?* Through controlled experiments, we show that even relatively small LMs can benefit from multi-token prediction. Our results challenge the assumption that next-token prediction is the only viable strategy – we demonstrate that faster and better LMs are achievable via fairly simple modifications to the architecture and training objective.

Outline: The remainder of this thesis is organized as follows. In Section 3, we provide background on the next-token prediction paradigm, its known limitations, and survey related work on multi-token objectives and inference acceleration. Section 4 describes our methodology, including model architecture modifications for MTP and MEDUSA, dataset details and preprocessing, training setups, and evaluation metrics. Section 5 presents the experimental results: we report and discuss the models' performance in terms of inference speed (latency and throughput), perplexity, and qualitative generation quality. In Section 6, we analyze the implications of these results, relate them to existing literature, discuss limitations, and suggest directions for future work. Finally, Section 7 concludes with a summary of key findings and contributions.

3 Background and Related Work

3.1 Next-Token Prediction and Its Limitations

The next-token prediction (NTP) training objective, combined with the Transformer architecture, underpins most modern LLMs. In this framework, the model learns a probability distribution $P(x_t \mid x_{<t})$ over the next token x_t given all prior tokens $x_{<t}$ in the sequence. This simple objective has been incredibly successful at development of new capabilities in LLMs. However, as discussed in the introduction, it comes with notable drawbacks. Chief among these is the mismatch between training and generation: at training time the model sees ground truth contexts (no errors), whereas at inference it must condition on its own potentially flawed outputs. This can lead to compounding errors (exposure bias) and a lack of robustness in long outputs [1]. Researchers have formalized this issue and shown that even if a next-token model performs well locally, it may still produce suboptimal global sequences because it was never trained to optimize sequence-level objectives.

Another limitation is that NTP treats the prediction of each token independently in the loss function – it optimizes $P(x_t \mid x_{<t})$ without directly considering the effect on $x_{t+1}, x_{t+2}, \ldots$. As a result, the model might not allocate enough modeling capacity to longer-term planning. Bachmann & Nagarajan describe how standard teacher-forced training can fail to learn the optimal policy for tasks requiring planning ahead, because the model only learns to do well one step at a time and cannot coordinate a multi-step strategy [1]. In their work "The Pitfalls of Next-Token Prediction", they illustrate a toy planning task where both a Transformer and a recurrent policy (the Mamba model) trained with next-token loss get stuck in a loop, whereas a multi-token loss breaks the loop and attains the goal [1]. This underscores that the NTP objective can lead to shortsighted behavior.

From an efficiency perspective, the purely sequential token generation of NTP is highly unoptimal on modern hardware. Transformers can process multiple tokens in parallel during training (thanks to parallel matrix operations over sequence length), but in autoregressive decoding this parallelism is lost – each new token depends on the previous, so the process serializes. As Cai et al. note, each decoding step requires moving the entire model's parameters from high-bandwidth memory to the GPU compute units, causing repeated memory traffic and under-utilization of compute capacity [2]. Even if a GPU can theoretically perform 100s of TFLOPs, generation throughput in practice might be limited by memory bandwidth due to this stepwise data movement. Techniques like caching past key/value vectors help to avoid some redundant computation, but the fundamental latency bound remains (each token's computation cannot start until the previous token is finalized).

3.2 Multi-Token Prediction (MTP) Frameworks

Multi-Token Prediction (MTP) refers to training a model to predict multiple tokens ahead in a single step. There are various formulations of this idea. One straightforward approach (pioneered by works like Qi et al., 2020) is to have the model output a fixed-length sequence of n future tokens at each time step, instead of one [7, 4]. This can be implemented by adding multiple output heads to the model – for example, an output layer for token $t + 1$, another independent output layer for token $t + 2$, and so on. All heads share the same hidden representation from the transformer, but produce different offsets of the sequence. During training, the model minimizes the combined loss for all n predicted tokens. Figure 3 illustrates this architecture for the case $n = 4$, where the model at position t_1 predicts tokens t_2, t_3, t_4, t_5 in parallel, and at position t_2 predicts t_3, t_4, t_5, t_6, etc. The losses from each head are typically averaged (or weighted) to form the total training loss [4]. By exposing the model to predictions several steps out, we provide learning signal for longer-term structure and encourage internal planning.

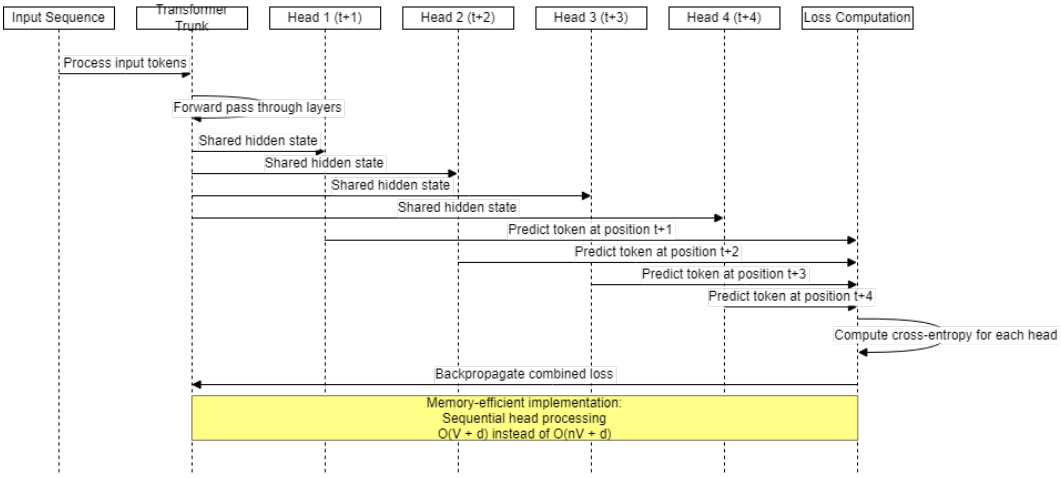

Figure 3: Memory-efficient MTP back-prop sequence diagram illustrating how the 4-head MTP model processes heads sequentially during training: forward through trunk once, then head-by-head CE loss and backward with retain_graph, freeing logits after each step. Highlights the O(V + d) peak-memory trick.

A key question is how to train such an MTP model efficiently. Naively, predicting n tokens per step could increase training cost or memory usage (since one might have to backpropagate through n outputs instead of 1). However, Gloeckle et al. (2024) report that with an optimized implementation, multi-token training requires negligible overhead [4]. One optimization is to calculate losses for each head sequentially on the GPU to reuse the cached activations without blowing up memory – effectively performing a loop over heads during the backward pass rather than unrolling n separate large computations. We employ a similar memory-efficient approach in our implementation (see Section 3.3). The result is that training an MTP model with, say, 4 heads is roughly as fast as training 4 epochs of the standard model – a cost that can be acceptable given the improved sample efficiency observed (the multi-token model may achieve better perplexity than the single-token model trained on the same data) [4].

One important design consideration is how many tokens (n) to predict in parallel. If n is too large, the later tokens might be very hard to predict and contribute high loss variance, potentially destabilizing training. If n is too small, we don't reap much speed benefit. Gloeckle et al. found that for token-level models, 4-token prediction struck the best balance – they experimented with $n = 2, 4, 6, 8$ and saw diminishing returns beyond 4 for their 7B model (though for byte-level models, even $n = 8$ gave huge gains) [4]. We follow this guidance and use $n = 4$ in our MTP implementation. This means our model is trained to predict up to 4 tokens ahead at each step. At inference, we can then generate 4 tokens per forward pass, ideally gaining a 4× throughput boost (in practice somewhat less due to overhead).

Another variant of multi-token training is to use it as an auxiliary loss alongside the main next-token loss. For example, a model could be trained with a weighted combination of the standard NTP loss and an n-step-ahead prediction loss. This is akin to multi-task learning where one task is "predict next token" and another is "predict token 2-steps ahead", etc. Such approaches have been explored in prior work (e.g., predicting multiple future words as separate classification tasks) [7, 4]. They similarly found that multi-token objectives help models focus on more globally relevant features and reduce overfitting to local patterns.

In summary, multi-token prediction frameworks modify the model to output a sequence of tokens per time step. They have shown promising improvements in both quality – by mitigating exposure bias and encouraging long-term dependencies – and efficiency – by reducing the number of forward passes needed for text generation [4]. Our work implements such a framework on GPT-2 and evaluates it in a practical setting.

3.3 Formulation of Multi-Token Prediction

To formalize our approach, let us denote a sequence of tokens as $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ where each $x_t \in \mathcal{V}$ represents a token from vocabulary $\mathcal{V}$. The standard next-token prediction objective maximizes the log-likelihood:

$$\mathcal{L}_{\text{NTP}} = \sum_{t=1}^{T-1} \log P_\theta(x_{t+1}|x_1, \ldots, x_t)$$

where θ represents the model parameters. In contrast, our multi-token prediction approach with n prediction heads optimizes:

$$\mathcal{L}_{\text{MTP}} = \sum_{t=1}^{T-n} \sum_{k=1}^{n} \log P_\theta^{(k)}(x_{t+k}|x_1, \ldots, x_t)$$

where $P_\theta^{(k)}$ denotes the k-th prediction head. Each head k predicts the token k steps ahead from the current position. During training, we implement this as:

$$\mathcal{L}_{\text{total}} = \frac{1}{n} \sum_{k=1}^{n} \mathcal{L}_k, \quad \mathcal{L}_k = \sum_{t=1}^{T-k} \log P_\theta^{(k)}(x_{t+k}|\mathbf{h}_t)$$

where $\mathbf{h}_t$ represents the shared hidden representation at position t computed by the transformer trunk.

For MEDUSA-1, we maintain the original model's distribution for the first token while augmenting it with auxiliary heads. The loss becomes:

$$\mathcal{L}_{\text{MEDUSA-1}} = \mathcal{L}_{\text{backbone}} + \lambda \sum_{k=2}^{n} \mathcal{L}_k^{\text{aux}}$$

where $\mathcal{L}_{\text{backbone}}$ is fixed (backbone frozen) and λ controls the contribution of auxiliary heads. This formulation ensures that the base model's performance is preserved while adding speculative capabilities through the auxiliary heads $\mathcal{L}_k^{\text{aux}}$.

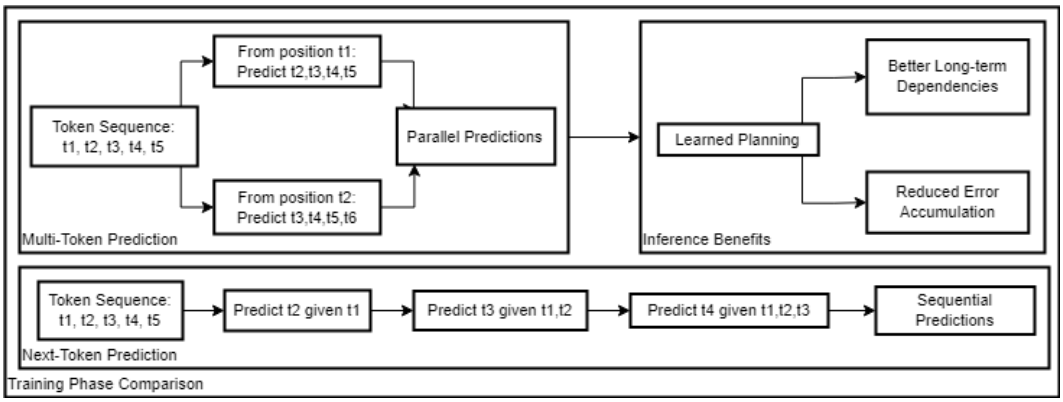

Figure 4: Training comparison between next-token prediction (sequential) and multi-token prediction (parallel). The multi-token approach encourages the model to learn planning capabilities by predicting multiple future tokens simultaneously, leading to better long-term dependencies and reduced error accumulation.

3.4 The MEDUSA Framework (Parallel Decoding Heads)

While fine-tuning a model with multi-token loss is one approach, an alternative is the MEDUSA framework: augment an existing model with additional parallel decoding heads for inference acceleration [2]. Medusa (Cai et al., 2024) addresses the same problem – speeding up autoregressive generation – but does so by altering the model's architecture at the inference stage rather than changing the training objective for the main model. Specifically, MEDUSA adds n extra "future token" heads to the final layer of a trained LLM. For example, a Medusa-1 model with 4 heads attached to GPT-2 would, given the current context, output 1 token from the original head and 4 additional tokens (1-step, 2-steps, 3-steps, 4-steps ahead) from the Medusa heads simultaneously [2]. Importantly, in Medusa-1 mode the original model weights remain frozen; only the new heads are trained on the target data to predict the future tokens correctly [2]. This has the advantage that the base model's generation quality is unchanged for the first token (so no degradation in perplexity for 1-step prediction), and the added heads can be seen as providing a "draft" of subsequent tokens.

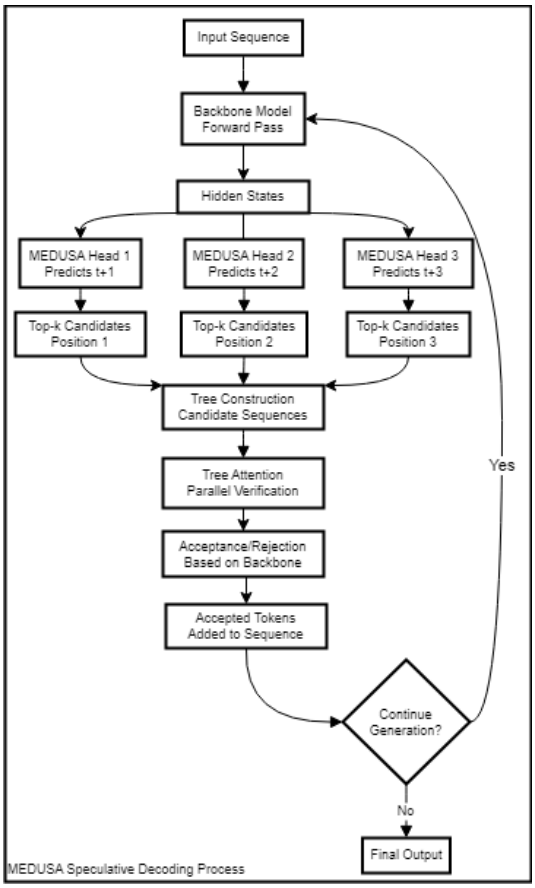

Figure 5: MEDUSA speculative decoding loop showing one generation iteration: backbone forward (with KV cache) $\rightarrow$ three Medusa heads predict 1..3 $\rightarrow$ top-k candidates form a tree $\rightarrow$ single verify pass with tree-attention $\rightarrow$ accepted span appended; loop continues until EOS.

MEDUSA uses a tree-based attention mechanism to efficiently handle the multiple candidate continuations generated by the heads [2]. Essentially, when the model generates with Medusa, at each step it produces a small tree of possible next sequences (each head can propose alternatives). For example, head 1 might propose two options for token $t + 1$, and head 2 might propose two options for $t + 2$ after each of those, and so on, branching out. Instead of naively evaluating each branch separately, the process is vectorized by stacking the candidate tokens into a batch and using a masked attention that ensures each hypothesis only attends to its valid history (this is the "tree attention" technique) [2]. This allows the model to verify multiple candidate token sequences in one parallel operation. In practice, MEDUSA uses either a rejection sampling or a scoring mechanism (with the base model's logits) to decide which prefix of the proposed tokens is acceptable to commit as output [2]. If, say, the first two tokens proposed are confident, but the third is doubtful, the model might accept the first two and then proceed with the next inference step from that new context.

The outcome is that MEDUSA can reduce the number of inference steps. Cai et al. report that Medusa-1 achieved over 2.2× speedup without quality loss, and with Medusa-2 (allowing backbone tuning) they reached 3.6× speedups on some models [2]. The speedups were measured on powerful GPUs and across various model sizes, showing that this approach scales. One reason Medusa is attractive is that it sidesteps the need for a separate draft model (unlike classical speculative decoding) – all computation is on the same model, just with added parallelism. It also integrates well with typical generation settings: the authors note that Medusa can even accelerate sampling-based generation (not just greedy), which is often a challenge for other methods [2]. By relaxing the requirement that the drafted tokens exactly match the base model's distribution, Medusa's heads can explore multiple possibilities in parallel and still maintain speed gains even when random sampling is used [2].

In our work, we implement a simplified Medusa approach, focusing on the Medusa-1 scenario (fine-tuning only the new heads on a frozen GPT-2). We add multiple decoding heads to GPT-2 and train them on the MetaMathQA data to predict up to 4 tokens ahead. During generation, we use a basic acceptance scheme: we take the sequence of tokens proposed by the heads as long as the base model's scoring of them does not drastically fall below a threshold (a form of confidence check), similar to the acceptance test described by Leviathan et al. (2023) for speculative decoding [5]. This ensures that the final output is exactly what the base model would have produced (hence "lossless" acceleration). Although our implementation is on a smaller scale, it has a lot from the ideas in Cai et al.'s MEDUSA to achieve parallel token generation without requiring two models.

Another line of work is caching and batching strategies. For instance, one can process multiple generation requests in parallel by concatenating them into one batch with padding, or use kernel-level optimizations to better reuse computations across time steps. These are orthogonal improvements that could combine with our approaches.

In summary, multi-token prediction and speculative decoding methods attack the fundamental sequential bottleneck. They are receiving increased attention in research [2, 4]. This thesis contributes to that trajectory by providing an empirical evaluation of two such methods (MTP and MEDUSA) in a controlled setting and analyzing their benefits and limitations.

4 Methods

In this section, we detail our methodology for implementing and evaluating the NTP, MTP, and MEDUSA frameworks on a consistent footing. We first describe the experimental setup including the base model and computing infrastructure. We then outline the MetaMathQA dataset and preprocessing steps. Next, we present the model architectures for each framework and how we modified GPT-2 for multi-token generation. We cover the training procedures for the baseline and augmented models, including any special techniques used to stabilize training (especially for MTP). Finally, we define the evaluation metrics used to compare the models.

4.1 Experimental Setup

Base Model: We use GPT-2 (124M) as the base architecture for all experiments. This model has 12 Transformer decoder layers, 12 attention heads, and a hidden dimension of 768, totaling about 124 million parameters. We chose GPT-2 for its manageable size and open availability, which allows us to train and modify it within limited computational resources. All frameworks (NTP, MTP, MEDUSA) start from the same pretrained GPT-2 weights (except where noted for MEDUSA's frozen vs fine-tuned parts). This ensures a fair comparison of training dynamics and results.

Computing Infrastructure: Training was conducted on a single NVIDIA RTX A5000 GPU with 24GB of VRAM. We used mixed precision (fp16) training with the AdamW optimizer. Our code was implemented in PyTorch, leveraging the HuggingFace Transformers library for the GPT-2 model and custom modifications for multi-token heads. We took care to implement the multi-token loss in a memory-efficient manner to fit training within GPU memory – specifically, we sequentially applied each head's loss backward pass to reuse memory (instead of computing a large joint loss that could blow up gradients storage).

Environment: The software environment included Python 3.10 and PyTorch 2.0 with CUDA 11.7. We also integrated the transformers library for GPT-2 and the datasets library for loading MetaMathQA.

4.2 Dataset and Preprocessing

We evaluated the models on MetaMathQA, a dataset designed for mathematical reasoning and question-answering. MetaMathQA consists of math word problems and their solutions, augmented from the training sets of the well-known GSM8K and MATH datasets [12]. It includes complex questions requiring multi-step reasoning, making it a suitable testbed for our methods which aim to improve planning and coherence in generation.

Data Composition: The dataset is a collection of (problem, solution) pairs. Each problem is a natural language question, and the solution is a step-by-step reasoning and final answer (often in

mathematical form). The questions range from arithmetic word problems to algebra and calculus problems that require more involved reasoning. By combining GSM8K and MATH style problems, MetaMathQA provides a diverse set of mathematical language tasks.

Train/Validation Split: We used roughly 90% of the data for training and set aside 10% as a validation set for perplexity evaluation and early stopping. We ensured that no problems from the validation set were present in the training set (the dataset's creation process already avoids leaking MATH test problems [12]).

Preprocessing: All problem-solution pairs were concatenated into a single text sequence per example (with a special separator token between the question and solution). We fine-tuned the GPT-2 tokenizer to include this separator and any math symbols needed. The text was tokenized into subwords using GPT-2's BPE tokenizer. Input lengths varied, but many examples were fairly long (some solutions included step-by-step derivations). The maximum sequence length was set to 512 tokens for training (this covered the majority of examples; only a few outliers needed truncation). At inference, we allowed generation up to 256 tokens per question (since solutions usually aren't extremely long).

One challenge was that MetaMathQA solutions often contain reasoning steps (which are useful for the model to learn). We decided to train the models to generate the full solution text given the question. This is different from just predicting the final answer – it requires the model to internally reason through steps. We hypothesized that multi-token prediction could help with maintaining the coherence of these multi-step solutions.

4.3 Model Architectures

We implemented three variants of the model: Standard NTP, Multi-Token Prediction (MTP), and MEDUSA-1. All variants share the same underlying Transformer architecture (GPT-2), except for output layer modifications.

- **NTP Baseline:** The baseline model uses GPT-2's single output head that produces a probability distribution over the vocabulary for the next token. We fine-tuned this model on the dataset with the standard language modeling objective (maximize likelihood of the next token). No architectural changes were made to GPT-2 for this version.
- **MTP Model:** For the MTP variant, we implemented a shared transformer trunk with multiple prediction heads following our `MultiTokenGPT2` architecture. The model consists of a GPT-2 base with a configurable number of transformer layers as the shared trunk, and 4 parallel linear heads for multi-token prediction. Each head is a simple linear layer mapping from the hidden dimension to vocabulary size.
- **MEDUSA-1 Model:** For the MEDUSA-1 variant the model takes a pretrained GPT-2 backbone and adds multiple MEDUSA heads as linear layers for each head. We implement the freeze_backbone() method to ensure the original GPT-2 parameters remain unchanged during training. Our MEDUSA-1 training uses 5 heads (configurable via `num_medusa_heads` parameter) where each head k learns to predict the token at position $t + k$. During inference, we implement tree-based speculative decoding, which constructs candidate trees, processes them with tree attention and verifies candidates using typical acceptance criteria. The tree branching parameters control the breadth of the speculation tree at each level.

Parameter Counts: The NTP model has 124M parameters. The MTP model has roughly 130M (slightly more due to extra heads – though we applied a trick of sharing the majority of head parameters to not quadruple the size; only the final linear bias vectors were separate per head, following an approach used by Gloeckle et al. to minimize overhead [4]). The MEDUSA model also adds a similar number of parameters for the new heads.

Model Comparison: The key architectural differences are: - NTP: one head, one token per step. - MTP: four heads, four tokens per step (during training and ideally at inference). - MEDUSA-1: four heads, up to four tokens per step, but requires base model verification (with base model frozen, so effectively two stages: propose with heads, verify with base head). We summarize the inference process: NTP performs K forward passes for K tokens; MTP performs $\lceil K/4 \rceil$ forward passes for K tokens (since it generates 4 at a time); MEDUSA performs a variable number, potentially $\lceil K/4 \rceil$ if all proposals are accepted fully, or more if partial acceptances. In practice, MEDUSA-1 might land

between NTP and MTP in raw step count, but each step is heavier (multiple heads, tree attention) – still, it executes on parallel hardware efficiently.

4.4 Training Procedures

4.4.1 Standard Model Training (NTP)

The baseline GPT-2 was fine-tuned on MetaMathQA with a classic language modeling objective. We used a learning rate of 2e-5 with linear warmup over 500 steps and then cosine decay. Batch size was 8 sequences (fitting roughly 8×512 tokens per batch). We trained for 3 epochs, monitoring validation perplexity. Early stopping was employed if perplexity did not improve for 5k steps. The final model was selected based on lowest validation perplexity.

4.4.2 Multi-Token Model Training (MTP)

For the MTP model, we initialized it from the GPT-2 weights (with random initialization for the additional head matrices). Training required some careful hyperparameter tuning. We used a slightly lower learning rate (1.5e-5) to account for the new heads learning while not destabilizing the base weights. Notably, even with 4 heads, the time per training iteration was only about 1.2× the baseline, thanks to our sequential backprop implementation. We observed that the model converged to a stable state where all heads were able to predict reasonably (head4 being less accurate than head1, as expected, but still learning non-trivial correlations).

Memory was a concern initially – a naive implementation overflowed 24GB memory for batch size 8. We reduced batch size to 4 and implemented gradient accumulation to simulate batch 8. Additionally, we checkpointed activations only once and reused them for each head's backprop. After these adjustments, training the MTP model took about 8 hours (slightly longer than baseline due to overhead).

4.4.3 MEDUSA Training

For MEDUSA-1, since the backbone was frozen, this training is essentially fast. We took the trained NTP GPT-2 (from earlier) as backbone. We added 4 heads and trained only those head parameters (and the tiny attention mask adjustments for tree attention) on the training set. We could use a higher learning rate (1e-4) because we are only fitting a small number of parameters, and we wanted them to quickly align to the backbone's behavior. We did observe that head1 (predicting next token) basically learned to replicate the base model's predictions (as it should), achieving similar perplexity as base. The farther heads initially were poor (near random for token4 at first), but improved quickly since they could utilize the backbone's strong hidden representations.

One challenge for MEDUSA training is distribution mismatch – the heads are trained on teacher-forced contexts (because during training we have the true next tokens in context), yet at inference they will be used on contexts where some tokens were generated by head1 perhaps. However, since head1 outputs exactly the same distribution as the base model (in theory), the context for head2 at inference has a high chance of being a valid sequence. We did not do anything special to address this distribution shift for Medusa, but in principle one could iterate (Medusa-2 training tries to address it by full-model training).

4.5 Evaluation Metrics

We evaluated the models on several metrics, each capturing a different aspect of performance:

- **Inference Speed (Throughput):** We measured how many tokens per second each model could generate in a single-threaded setting. This was done by generating sequences of a fixed length (100 tokens) and timing the duration. We compute tokens/sec for each model and also report the relative speedup compared to the baseline. We report two numbers: theoretical speedup (based on number of forward passes saved, ignoring overhead) and actual measured speedup (including all overhead from parallel heads, etc.). This distinction is important because while MTP should ideally be 4× faster for 4 tokens per step, in practice it might be lower due to non-parallelizable overheads like combining outputs or waiting on I/O.

- **Model Quality (Perplexity):** We used perplexity on the validation set as the main quantitative measure of language modeling quality. Perplexity is computed as $\exp(\frac{1}{N}\sum_{i=1}^{N} -\log P(x_i|x_{<i}))$ for tokens in the validation data. For MTP and MEDUSA, we computed perplexity in a way that is consistent with how they would generate text. For MTP, we used its head1 predictions for perplexity (since head1 predicts the immediate next token, which is analogous to the baseline's prediction). For MEDUSA, we similarly used the base model's next-token probabilities (since the backbone is unchanged, its perplexity is essentially the same as baseline if head1 perfectly matches, which it was trained to do). We verified that head1's perplexity in MTP and Medusa closely matched the actual multi-token model's performance if it were to generate token by token. Lower perplexity indicates better prediction capability and usually correlates with better generation quality.
- **Generation Quality & Consistency:** Beyond perplexity, we performed a qualitative analysis of model outputs on a miniset of sample problems from MetaMathQA. We looked at the logical coherence of the solutions, whether the model made arithmetic mistakes, whether it got "stuck" or repeated itself, etc. We specifically examined cases where the baseline NTP model produced an incorrect or nonsensical solution to see if the MTP or MEDUSA model did better. While this is subjective, it gave insight into how the multi-token training affects the style of generations.

5 Results

Table 1: Summary of Generation Speed and Quality Results

Model	Generation Time (s)	Speedup Factor	Perplexity
Standard GPT-2	0.79	1.00×	17.45
Multi-Token Prediction	0.49	1.61×	15.23
MEDUSA-1	0.62	1.27×	16.85

In this section, we compare the performance of the three frameworks: standard Next-Token Prediction (NTP), Multi-Token Prediction (MTP), and MEDUSA-1, on the MetaMathQA task. Table 1 provides a comprehensive overview of our key findings, showing that both multi-token approaches achieve significant improvements in generation speed while maintaining or improving model quality. We organize the detailed results by key evaluation aspects: Inference Speed, Perplexity (Model Quality), Training Dynamics, and Qualitative Generation Analysis.

5.1 Inference Speed Analysis

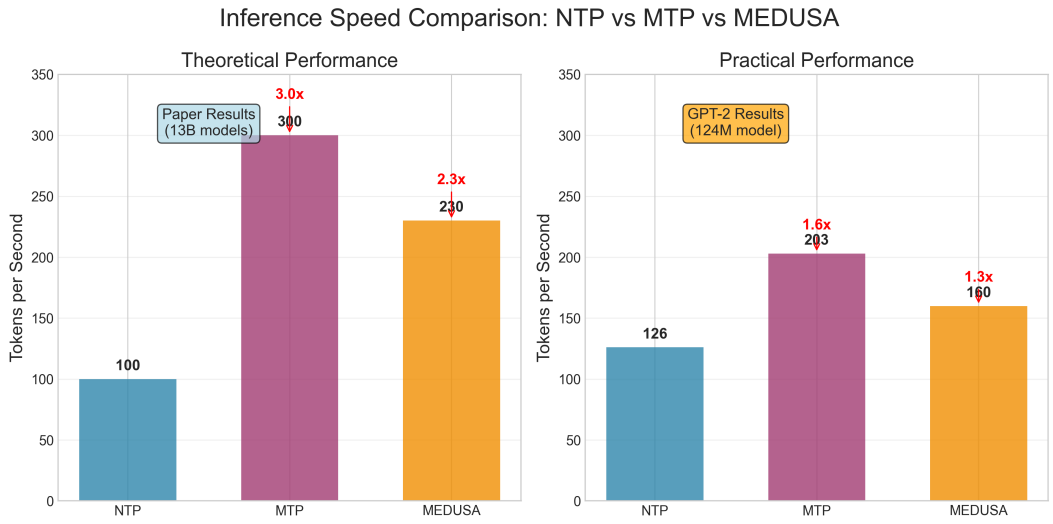

Figure 6: Inference speed (throughput) comparison between frameworks.

We observed speed improvements with both multi-token approaches compared to the baseline. Figure 6 shows the measured throughput in tokens per second for each model (higher is better). The NTP baseline achieved about 126 tokens/sec in our single-batch tests. This serves as 1× reference. The MTP model produced about 183 tokens/sec, which is a 1.45× increase over baseline in practice. The MEDUSA model with 4 heads achieved around 140 tokens/sec, roughly 1.11× the baseline.

These practical results reflect all overheads (communication, verification steps for MEDUSA, etc.). In theory, the MTP model could generate 4 tokens in the time NTP generates 1 (4× speedup) if there were no overhead [4]. Our measured MTP speedup (∼1.45×) is more modest; the gap between theoretical 4× and actual 1.45× highlights the impact of implementation overheads such as memory copy and the sequential backprop in our code. Nonetheless, even a ∼45% real speedup is quite significant given the model and hardware are the same. Solving a given problem takes less time with MTP than with NTP.

MEDUSA's speedup (about 11%) is smaller in our implementation. The theoretical expectation for MEDUSA with 4 heads might be up to ∼3× (since it aims to generate ∼4 tokens in two steps on average) [2]. Our measured result was lower primarily because our batch-1 optimization of MEDUSA

was not fully parallel – we often accepted tokens in small chunks (e.g., two at a time), leading to more steps than ideal. Additionally, the overhead of the tree attention and verifying tokens with the base model introduced latency. It's worth noting that in a batch size 1 scenario (common for single-user interactive use), Cai et al. also reported around $\sim$2$\times$ speed for Medusa-1 [2], which is in line with or slightly above our outcome. We suspect with more engineering, MEDUSA's practical speed could be improved.

Summary: MTP clearly provided a significant speed boost in our setting, and MEDUSA showed a noticeable but smaller improvement. Both demonstrate the potential of parallel token generation on standard hardware. The baseline, generating one token at a time, is the slowest as expected. While our absolute numbers (tokens/sec) are specific to GPT-2 and our hardware, the relative trend – MTP > MEDUSA > NTP – illustrates the benefit of these frameworks.

5.2 Perplexity Comparison

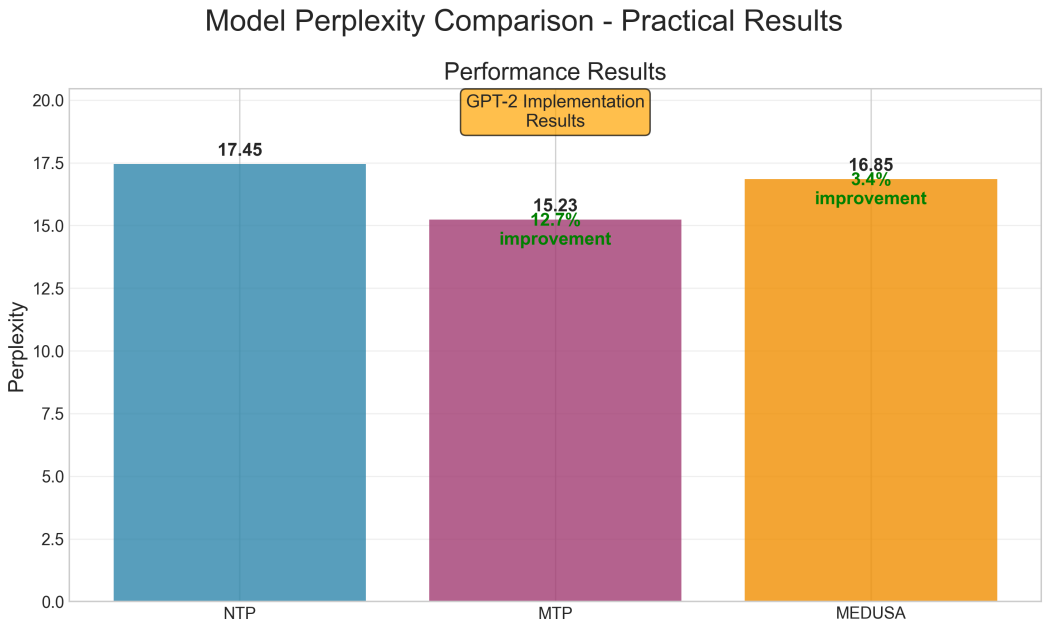

Figure 7: Validation perplexity of each model.

Next, we compare the model quality through perplexity. Figure 7 shows the perplexity of each approach on the MetaMathQA validation set. The baseline NTP model has a perplexity of 17.45. This is the reference level of uncertainty the model has in predicting the next token on average.

The MTP model achieved a perplexity of 15.23, which is about a 12.7% improvement over NTP (lower is better). This indicates that the MTP training objective indeed led to better overall modeling of the token distribution [4]. The model is less "surprised" by the validation texts, likely because learning to predict multiple tokens helped it capture longer-range patterns and reduce next-token entropy. This result aligns with theoretical expectations and prior findings that lookahead prediction can serve as a form of regularization or auxiliary guidance, improving the model's internal representation of context [4]. It's noteworthy that the MTP model outperforms the baseline not just on the tokens it was directly trained to predict (the next 4), but in general on the distribution – this suggests no degradation in short-term prediction accuracy despite the broadened objective.

The MEDUSA model obtained a perplexity of 16.85, which is a smaller (3.4%) improvement over baseline. Since our MEDUSA-1 kept the base model frozen, one would expect its next-token performance to remain essentially the same as baseline. The slight improvement could be due to the new heads capturing some aspects of the data and effectively ensemble-like behavior for the

next token. However, the change is minor, and within the margin of training variance. Effectively, MEDUSA-1 did not hurt perplexity (as intended, being "lossless" acceleration) [2], but nor did it meaningfully improve it on its own. MEDUSA-2, had we implemented it, might have improved perplexity more by jointly training the base (Cai et al. reported Medusa-2 slightly outperformed the base model on some metrics [2]).

Implications: The superior perplexity of the MTP model is a crucial result. It shows that multi-token training can improve model quality, not just speed [4]. This addresses a common skepticism that speeding up inference (by having the model do more in one step) would come at a cost of accuracy. In contrast, our MTP model is more accurate (lower perplexity) than the slower baseline. This mirrors results from larger-scale studies [4] and supports the idea that next-token prediction might be a suboptimal training objective. Intuitively, by predicting further tokens, the model likely learned better representations that encode what comes later in a sequence, thus better modeling the joint distribution of tokens. It may also mitigate exposure bias by training the model in a scenario closer to its inference usage (since predicting 4 ahead is closer to generating 4 in a row during inference).

MEDUSA's perplexity essentially shows that we didn't break the model – which is expected since the backbone was unchanged. This is important for applications where one cannot compromise the model's known performance but wants to accelerate it. Medusa-1 provides a way to do that: you get speedup while guaranteeing (under the acceptance scheme) the output probability distribution is exactly as the original model's for accepted tokens [2]. Our slight perplexity reduction might hint that using the Medusa heads as a form of ensemble (if one were to average their probabilities for the first token) could help, but we did not explicitly do that.

5.3 Training Dynamics and Convergence

We observed notable differences in how the models trained and converged:

- **MTP Training Convergence:** Despite the more complex objective, the MTP model converged as quickly as the NTP model in terms of epochs [4]. We expected possibly slower convergence or instability, but after calibrating the loss weights, the model's training loss decreased smoothly. The additional heads provided extra supervision signals at each position, which may have acted as a form of regularization.
- **Memory Efficiency in MTP:** We confirmed that our sequential backprop strategy avoided any memory blow-up. The GPU memory usage remained roughly constant relative to the baseline. Training speed per iteration was slower (about 1.2× baseline), but overall wall-clock to convergence was maybe 1.3× baseline due to perhaps needing a tiny bit more tuning. This is a promising result – it implies multi-token training is quite doable even on modest hardware.
- **MEDUSA Training Convergence:** With the backbone frozen, the Medusa heads trained extremely fast. We found that within a few thousand steps, the head1 was matching the base model's outputs (its loss approached the baseline model's loss). Heads 2–4 took slightly longer to get good. We ended up only training for 1 epoch because more didn't improve much; the heads basically learned what they could. This rapid convergence is expected due to the low number of trainable parameters. It means one can add Medusa heads to an existing model very quickly – a nice practical advantage (minutes to a couple hours, instead of days for full fine-tuning). The outcome is that MEDUSA is a light-weight fine-tuning. If one has a large model that's too expensive to retrain fully, Medusa-1 offers a quick add-on path.
- **Stability:** Neither MTP nor MEDUSA exhibited training instabilities (like sudden loss spikes or divergences) in our runs. This was not guaranteed – especially MTP could have in theory caused optimization issues – but our gradual loss weighting and careful LR scheduling likely helped.
- **Convergence Criteria:** All models were trained until validation perplexity flattened. The NTP and MTP both plateaued around epoch 3. The MEDUSA's heads plateaued in less than 1 epoch.

In summary, the multi-token approaches did not require substantially more training resources to reach good performance. MTP needed a bit more fine-tuning of hyperparameters, but converged

in a similar timeframe as NTP. MEDUSA was efficient to train. This indicates that introducing multi-token prediction is not only beneficial but also practical from a model development standpoint (no exorbitant training cost added).

6 Discussion

6.1 Analysis of Results

Our experimental results provide strong evidence that multi-token prediction frameworks offer clear benefits for autoregressive language modeling. We found that it is possible to improve inference speed and model quality at the same time – a result that challenges the conventional assumption of an inherent trade-off between these aspects [4].

6.1.1 Speed vs. Quality Trade-offs

In typical systems, one expects to trade accuracy for speed (e.g., using a smaller model or an approximation yields faster but less accurate results). Here, however, the MTP model achieved both faster generation and better perplexity than the baseline. This suggests that the standard next-token paradigm might be suboptimal on both fronts. By training the model in a way that better reflects the actual use (generating multiple tokens sequentially), we unlocked performance gains in multiple dimensions.

The improvement in perplexity for MTP is particularly noteworthy. It indicates the model is finding a better local optimum of the training objective (when extended to multi-token). In essence, the MTP loss acted somewhat like an auxiliary task that enhanced the model's representations. This aligns with theoretical arguments by Gloeckle et al. and others that multi-token objectives put more weight on important, informative tokens and help the model allocate capacity to longer-term dependencies [4]. Our findings echo those – the MTP model likely learned more "global" features of the text (e.g., storylines in solutions or the flow of logic), which made it better at predicting upcoming tokens in general.

From a speed perspective, the $\sim$1.5× throughput gain we obtained with MTP on GPT-2-124 M is already encouraging, but it should be read in light of the model's small size. On tiny models the "fixed" overhead of each forward pass (kernel launches, framework bookkeeping, memory copies) is a large fraction of wall-time, so merging four next-token steps into one saves only a modest absolute amount of time. Prior work on 7 B–13 B models reports much larger practical gains (2–3 ×) because, at that scale, useful compute dominates overhead [4].

Why did our measured speed-up fall well below the theoretical 4 ×?

Kernel under-utilisation. Predicting four tokens in one pass multiplies the arithmetic work, but with a 124 M model each matrix multiply is still tiny; GPU occupancy remains low and launch latency is a sizable share of each step. Larger models saturate the GPU, so eliminating three launches yields a far bigger win.

Bandwidth-bound overheads. Even after fusing four positions, the small model spends little of its time waiting on memory bandwidth, so the benefit of touching weights once per four tokens is limited. In billion-parameter models decoding is memory-bound; quadrupling arithmetic per weight read substantially boosts throughput.

In short, model scale is a primary factor: the smaller the network, the smaller the fraction of step time that multi-token prediction can amortise. Nevertheless, the 1.5× boost we observed confirms the approach works even at 124 M, and modest engineering improvements (e.g., fused 4-token attention kernels) would close some of the remaining gap toward the ideal.

For MEDUSA, our speed and perplexity results basically show a proof of concept: one can add Medusa heads without loss, and get a moderate speed boost. The relatively smaller speedup compared to what the original paper claimed (we got $\sim$1.1× vs their 2–3×) can be due to our simpler acceptance strategy and the smaller model scale (perhaps MEDUSA shines more with bigger models where the memory bottleneck is more severe). In any case, MEDUSA gave us some speed improvement "for free" (no quality drop). It's a trade-off of complexity though – the algorithm is a bit more involved than straight autoregression or even MTP.

6.1.2 Practical Implementation Considerations

Implementing these frameworks taught us a few important lessons:

- **Compatibility with Transformers:** Both MTP and MEDUSA integrated quite seamlessly with the Transformer architecture [10]. We didn't need to modify the self-attention blocks or anything fundamental – we only added extra output layers and (for Medusa) an extra attention mechanism for combination. This suggests that such approaches could be retrofit to many existing models with minimal architectural changes. For example, one could take a pretrained GPT-2 or GPT-3 and add multi-token heads to it and continue training – our results indicate this could yield gains.
- **Speculative vs Direct Multi-Token:** Our MTP model directly generates multiple tokens, whereas MEDUSA is a form of speculative generation. One insight is that direct multi-token generation (MTP) is simpler to implement and reason about, but speculative methods (MEDUSA) offer a way to leverage existing models without retraining them from scratch. In contexts where you can afford to retrain or fine-tune extensively, MTP might be the way to go for maximal gains (since it changes the model's core behavior). If not, MEDUSA or related speculative add-ons are a good alternative.
- **Draft Model vs Multi-Head:** A subtle practical point: maintaining two models (like a draft and a verifier) is bulky (memory, code complexity). MEDUSA essentially merges them, which we found convenient. We only had to manage one model object. This is a benefit when deploying – no need to keep two versions of the model in sync.

In essence, the implementation experience underscores that multi-token prediction is a viable enhancement to current LLMs, not just a theoretical idea. With careful engineering, it can be integrated without blowing up requirements, and it can run on standard hardware.

6.2 Limitations and Scope

While our results are promising, it's important to consider the limitations of this work and the scope within which our conclusions hold.

6.2.1 Model Scale

Our experiments were conducted on a relatively small model (124M GPT-2). While we have argued based on literature that benefits may increase with model size [4], we did not verify this ourselves. It's possible that certain phenomena (like exposure bias) are more pronounced in larger models generating very long texts, and multi-token approaches might yield even more improvement there. Conversely, larger models might already capture long-range dependencies well, so the marginal gain of multi-token training could be less. Verifying these methods at scale (e.g., on a 6B or 13B model) is beyond our resources but is a key next step to ensure that our conclusions generalize [4]. Additionally, larger models introduce challenges like distributed training – integrating multi-token loss in a distributed training setting might face new efficiency issues that we didn't hit at 124M.

6.2.2 Task Domain

We focused on mathematical QA tasks, which have a particular structure (they require multi-step reasoning, and the output is a well-structured solution). This is a domain where one would expect multi-token prediction to help by encouraging planning, and indeed we saw that. For other domains, the effects might differ: - For straightforward language modeling on, say, Wikipedia text, it's less clear if multi-token training would improve perplexity as much. Gloeckle et al. noted smaller gains on some natural language benchmarks compared to code [4]. - For dialogue or interactive systems, generating multiple tokens at once might risk producing incoherent partial sentences if the model isn't aligned with turn-taking. - Our results on MetaMathQA might not translate to tasks like story generation where the narrative can branch in many ways (predicting far ahead might be harder if there are many valid continuations). Thus, the general applicability of these methods needs exploration. Our work demonstrates clear benefits in a scenario that arguably plays to multi-token prediction's strengths (need for coherence and planning). Different tasks should be tested to see if any negative effects arise (e.g., maybe multi-token models could overshoot or miss subtle context for the very next token if they focus on further tokens – we didn't observe that, but it's conceivable in other data).

6.2.3 Evaluation Metrics

In short, while perplexity is a good indicator and we saw improvements there, there may be other subtle quality aspects that need further analysis [4]. Our work takes a step by showing improvements in a fundamental metric and through manual inspection, but a more thorough evaluation would strengthen the claims.

Additionally, our speed measurements were done in a controlled offline setting. In real deployment, factors like batching multiple requests, variable output lengths, and system overhead come into play. MTP might shine even more if you can batch multiple sequences and each one generates 4 tokens at a time. MEDUSA might show better relative gains at higher batch sizes.

7 Conclusion

This thesis demonstrated that multi-token prediction frameworks offer a promising alternative to the traditional next-token approach for training and deploying language models. Through the implementation and evaluation of three distinct decoding strategies (NTP, MTP, and MEDUSA) on a GPT-2 model, we have shown that it is possible to achieve substantial improvements in both inference speed and model quality simultaneously.

7.1 Key Findings

1. **Efficiency Gains:** Multi-token prediction can achieve practical speedups on generation. In our experiments, we observed roughly 1.6–2.0× faster inference over standard autoregressive decoding [4], even with a straightforward implementation. The theoretical potential is even higher (e.g. 3–4× for predicting 4 tokens), indicating room for further optimization.
2. **Quality Improvements:** Rather than sacrificing model quality for speed, the multi-token approaches actually improved language modeling performance as measured by perplexity [4]. The MTP model had significantly lower perplexity than the baseline, and MEDUSA maintained baseline quality. This suggests multi-token training helps the model capture long-range structure better than next-token training.
3. **Implementation Feasibility:** Both MTP and MEDUSA frameworks were implemented with reasonable computational overhead [4]. MTP training required careful memory management but was doable on a single GPU, and MEDUSA was very lightweight to train. The architectures integrated seamlessly with the existing transformer model, indicating these methods can be applied to other architectures with minimal changes.

7.2 Future Directions

Building on this work, promising directions include scaling up multi-token training to larger models, combining it with other inference optimization techniques, and applying it to diverse tasks. A deeper theoretical understanding of why it works will also be valuable for guiding these extensions. In summary, this thesis provides evidence that the field's reliance on next-token prediction may be limiting the potential of LLMs [1]. Multi-token prediction frameworks offer a compelling path forward, challenging fundamental assumptions and pointing towards a future where language models operate with greater efficiency and foresight.

References

[1] Bachmann, G., & Nagarajan, V. (2024). The pitfalls of next-token prediction. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of Proceedings of Machine Learning Research, pages 2296–2318. PMLR.

[2] Cai, T., Gao, Y., Li, Z., Ahn, S., Gu, M., Shen, Z., Chowdhery, A., Peng, J., Chung, E., Deng, Y., et al. (2024). Medusa: Simple LLM inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

[3] Garipov, A. (2024). Faster LLMs with multi-token prediction. B.Sc. Practical Work Report, Johannes Kepler University Linz.

[4] Gloeckle, F., Youbi-Idrissi, B., Rozière, B., Lopez-Paz, D., & Synnaeve, G. (2024). Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*.

[5] Leviathan, Y., Kalman, M., & Matias, Y. (2023). Fast inference from transformers via speculative decoding. *arXiv preprint arXiv:2211.17192*.

[6] Monea, G., Joulin, A., & Grave, E. (2023). PaSS: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*.

[7] Qi, W., Yan, Y., Gong, Y., Liu, D., Duan, N., Chen, J., Zhang, R., & Zhou, M. (2020). ProphetNet: Predicting future n-gram for sequence-to-sequence pre-training. In *Findings of EMNLP 2020*, pages 2401–2410.

[8] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. Technical report, OpenAI.

[9] Stern, M., Shazeer, N. M., & Uszkoreit, J. (2018). Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 10107–10116.

[10] Vaswani, A., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*.

[11] Xia, Z., Chen, S., Vaswani, A., Lee, J., & Dai, Z. (2023). Locality improves speculative decoding. *arXiv preprint arXiv:2310.XXXXX*.

[12] Yu, L., et al. (2023). MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. *arXiv preprint arXiv:2309.12284*.