Breaking the Autoregressive Chain: Hyper-Parallel Decoding for Efficient LLM-Based Attribute Value Extraction

Theodore Glavas[*1,2,3,4], **Nikhita Vedula**[1], **Dushyanta Dhyani**[1], **Yilun Zhu**[†1], **Shervin Malmasi**[1]
[1]Amazon.com, Inc., [2]McGill University,
[3]Mila - Quebec AI Institute, [4]Int. Lab. Learning Systems
theodore.glavas@mail.mcgill.ca, yz565@georgetown.edu, {veduln, dhyanidd, malmasi}@amazon.com

Abstract

Some text generation tasks, such as Attribute Value Extraction (AVE), require decoding multiple independent sequences from the same document context. While standard autoregressive decoding is slow due to its sequential nature, the independence between output sequences offers an opportunity for parallelism. We present Hyper-Parallel Decoding, a novel decoding algorithm that accelerates offline decoding by leveraging both shared memory and computation across batches. HPD enables out-of-order token generation through position ID manipulation, significantly improving efficiency. Experiments on AVE show that attribute-value pairs are conditionally independent, enabling us to parallelize value generation within each prompt. By further stacking multiple documents within a single prompt, we can decode in parallel up to 96 tokens per prompt. HPD works with all LLMs, and reduces both inference costs and total inference time by up to 13.8X without compromising output quality, potentially saving hundreds of thousands of dollars on industry AVE tasks. Although designed for attribute extraction, HPD makes no assumptions unique to the AVE domain and can in theory be applied to other scenarios with independent output structures.

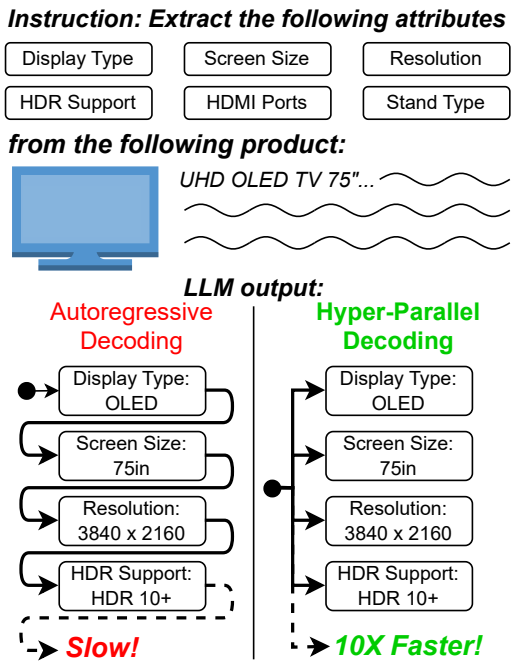


Figure 1: AVE on the e-commerce domain: Given a set of attributes for the category *Television*, values for each attribute are autoregressively generated using an LLM. Our method, **Hyper-Parallel Decoding (HPD)** parallelizes the extraction of values within the prompt to dramatically increase throughput and decrease cost.

1 Introduction

Information Extraction (IE) is a broad subcategory of Natural Language Processing (NLP) tasks that extract structured information from unstructured data. Generative IE has emerged from applying LLMs to IE in domains like e-commerce, medicine and real estate (Brinkmann et al., 2024b; Agrawal et al., 2022; Kvet et al., 2025), and recent work has shown that LLM-based IE is highly effective (Roy et al., 2024). However, the computational cost of generative LLMs limits their adoption in real-world scenarios (Zhang et al., 2025). Prior work has proposed several techniques such as quantization, pruning and distillation (Zhou et al., 2024) to effectively accelerate LLM inference, but inference costs can still be prohibitive on large-scale data. Moreover, these approaches are fundamentally limited by the autoregressive decoding process, which inherently limits the potential for parallelization on efficient hardware such as GPUs. This motivates our central research question: ***How can we overcome the autoregressive bottleneck to dramatically increase the speed of LLM-based IE and reduce cost, while maintaining output quality?***

[*] This work was done as part of an internship at Amazon.
[†] Work done while at Amazon. Currently at Apple.

We focus on Attribute Value Extraction (AVE), a sub-task of IE. AVE requires extracting values for multiple predefined attributes from a single document (see Figure 1). Since AVE often needs to be performed on massive sets of documents, it can greatly benefit from offline batched inference. Specifically, e-commerce is a popular domain for AVE as product listings often contain noisy, unstructured text. Crucial applications such as search (Lu et al., 2021; Xiao et al., 2021), recommendation (Hwangbo et al., 2018; Truong et al., 2022) and question answering (Zhang et al., 2020; Rozen et al., 2021) can greatly benefit from *structured* information in the form of attribute-value pairs.

Autoregressive decoding is slow due to its sequential nature (Shazeer, 2019). However, for tasks like AVE, outputs are independent and can be decoded in parallel. Prior literature has explored the idea of decomposing LLM generation into parallel components to accelerate decoding speed. For example, Ning et al. (2024) show that independent reasoning chains can be decoded in parallel for problem solving. Although single-batch latency is reduced, such methods do not translate to the offline setting where batched inference is used, and where throughput and cost are the target metrics. Approaches such as Medusa (Cai et al., 2024) support batched inference and can increase throughput by proposing a chain of multiple consecutive tokens in parallel. However, as these tokens are dependent on each other, throughput is limited by the maximum length of the proposals and the verification step required to ensure quality.

To address the above challenges, we introduce **Hyper-Parallel Decoding (HPD)**, a novel inference method that can increase the parallelization of generative AVE, unlocking extreme throughput gains up to 13X. Our key insight is that different generative tasks (e.g. attribute values) can be treated as *conditionally independent* given a common document context. HPD leverages this inherent structure of AVE to *break the autoregressive dependency* during generation, and simultaneously generate multiple attribute-value pairs. We further introduce an additional parallelization dimension by also extracting attributes of multiple documents within a shared prompt in parallel, analogous to how CPU hyper-threading can simultaneously complete multiple instructions. Our findings demonstrate that generating non-consecutive, independent tokens in parallel is far more efficient for increasing throughput than prior work. Although we evaluate our approach on e-commerce datasets, HPD makes no assumption exclusive to this setting and can in theory be applied to any domain where the conditional independence of the LLM output holds. We make our code available.[1] To summarize, our main contributions are:

[1] We introduce Hyper-Parallel Decoding, a novel, high speed inference method that can increase the parallelization of attribute-value pair extraction to generate up to 96 tokens per prompt.

[2] We demonstrate compatibility with a variety of other cost-reduction techniques including quantization, knowledge distillation and batched inference, enabling strong performance in the cost-constrained offline inference setting.

[3] We demonstrate the effectiveness of HPD on three e-commerce AVE benchmark datasets, resulting in no quality drop and achieving upwards of 13X execution time reduction and 13X monetary cost reduction for select LLMs.

2 Related Work

AVE Early work in AVE formulated the problem as a sequence labeling task, identifying spans of the input document as attribute values. Tagging was performed using specialized domain rules (Zhang et al., 2009; Vandic et al., 2012) and later generalized using BiLSTM-CRF (Huang et al., 2015) and BERT-based (Xu et al., 2019a) architectures. AVE was later formulated as a question answering (QA) task, with AVEQA (Wang et al., 2020) and MAVEQA (Yang et al., 2022) using BERT (Devlin et al., 2019) and ETC (Ainslie et al., 2020) encoders. Both tagging and QA-based AVE rely on the existence of a comprehensive labeled training set, and struggle to generalize to unseen/implicit attributes and values. The generative IE paradigm (Blume et al., 2023; Roy et al., 2024) takes advantage of LLMs to offer superior performance in AVE tasks with few labels or high data sparsity, which is common in the e-commerce domain. Recently, ExtractGPT (Brinkmann et al., 2024b) has shown that generating attribute-value pairs with LLMs for product AVE is highly effective (Brinkmann et al., 2024a; Shinzato et al., 2023; Sabeh et al., 2024).

Parallel LLM Inference We limit our review to works which increase LLM inference parallelism,

[1] https://github.com/networkslab/HPD

as other efficient LLM methods are largely orthogonal to our work. Research in LLM inference parallelization often aims to reduce *latency* in an online setting, where requests are streamed one at a time. Skeleton-of-Thought (Ning et al., 2024) breaks down complex reasoning problems into a set of parallel sub-tasks, which can be simultaneously decoded via API calls to an LLM service. This takes advantage of the unused batch dimension in online inference, but does not translate to the offline setting where we assume GPU VRAM to already be saturated by batching independent prompts. APAR (Liu et al., 2024) partially addresses this issue using paged-attention to share part of the common key-value cache across batch elements. However, the attention operation must still be repeated for every parallel stream, unlike our method which shares both the memory and computation within a single batch element.

Another common paradigm is to draft a chain of multiple *consecutive* tokens in parallel, usually followed by a verification step (Fu et al., 2024; Lin et al., 2025; Cai et al., 2024). In this scenario, the level of attainable parallelism is limited by the quality of the token chain, which degrades for longer chains. Speculative decoding (Leviathan et al., 2023) is similarly limited. In practice, the ratio of generated tokens to inference steps ranges from 2 to 5, as erroneous tokens must either be rejected by the verification step or contribute to quality degradation.

3 Attribute-Value Extraction (AVE) Task

Our input is a set of documents $\boldsymbol{x}_i \in \mathcal{X}$, where $\boldsymbol{x}_i = \{x_1, x_2, \dots\}$ is represented as a sequence of tokens in the vocabulary space $\mathcal{Y}$. The documents are partitioned into categories $C(\boldsymbol{x}_i) = c \in \mathcal{C}$. Each category c has a list of predefined attributes $\boldsymbol{a}_{1:N} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \dots, \boldsymbol{a}_N\}$, where each $\boldsymbol{a}_n \in \mathcal{A}$, $\boldsymbol{a}_n = \{a_1, a_2, \dots\}$ is also represented as a sequence of vocabulary tokens. Note that we drop the indexing in c to simplify the notation. The goal of AVE is to extract attribute values $\boldsymbol{v}_n \in \mathcal{V}$, $\boldsymbol{v}_n = \{v_1, v_2, \dots\}$ associated with each attribute for all products in $\mathcal{X}$. Figure 1 illustrates the AVE task for a product in the category *Television*.

In a generative LLM setting, we first construct a prompt $\boldsymbol{p}(\boldsymbol{x}_i, \boldsymbol{a}_{1:N})$, as shown in Figure 2. The prompt contains instructions for the LLM, the attributes to extract and the input context. For each prompt, we wish to generate an output $\boldsymbol{y} = \{(\boldsymbol{a}_1, \boldsymbol{v}_1), (\boldsymbol{a}_2, \boldsymbol{v}_2), \dots, (\boldsymbol{a}_N, \boldsymbol{v}_N)\}$ containing the attribute/value pairs extracted from the input data. In practice, $\boldsymbol{y}$ is a sequence of tokens $\{y_1, y_2, \dots\}$ consisting of the concatenated attributes/values in a structured text format. We use JSON to represent the text (Figure 2 (a)), but the approach is format-agnostic. The full prompt appears in Appendix E.

During inference step $t \in \{1, 2, \dots\}$, we provide the LLM a sequence consisting of the prompt $\boldsymbol{p}(\cdot)$ concatenated with the partial output $\boldsymbol{y}_{1:t-1}$ if $t > 1$, which we shorten to $\boldsymbol{y}_{<t}$. We define our decoder LLM network $F(\cdot)$ to output raw logit values over the vocabulary space $\mathcal{Y}$. Using autoregressive generation, we sample the next token from the random variable $Y_t \in \mathcal{Y}$. The conditional PDF of Y_t is defined as the predicted probability distribution

$$\hat{P}(Y_t = y \mid \boldsymbol{p}, \boldsymbol{y}_{<t},) = \sigma(F(\boldsymbol{y}_{<t}, \boldsymbol{p})) \in [0,1]^{|\mathcal{Y}|}, \tag{1}$$

where $\sigma(\cdot)$ is the softmax function. Upon completion of the generation process after step $t = T$, we parse the generated tokens $\boldsymbol{y}_{1:T}$ to obtain the attribute-value pairs.

Metrics We evaluate AVE on the below metrics:

(i) **Throughput:** rate of products processed (product/s) on average per GPU with a benchmark server using wall-clock time.

(ii) **Cost:** GPU rental cost per 1K products processed ($ / 1k products), or API cost for proprietary LLMs.

(iii) **Output Quality:** F1 score of values against ground truth or pseudo-ground truth labels.

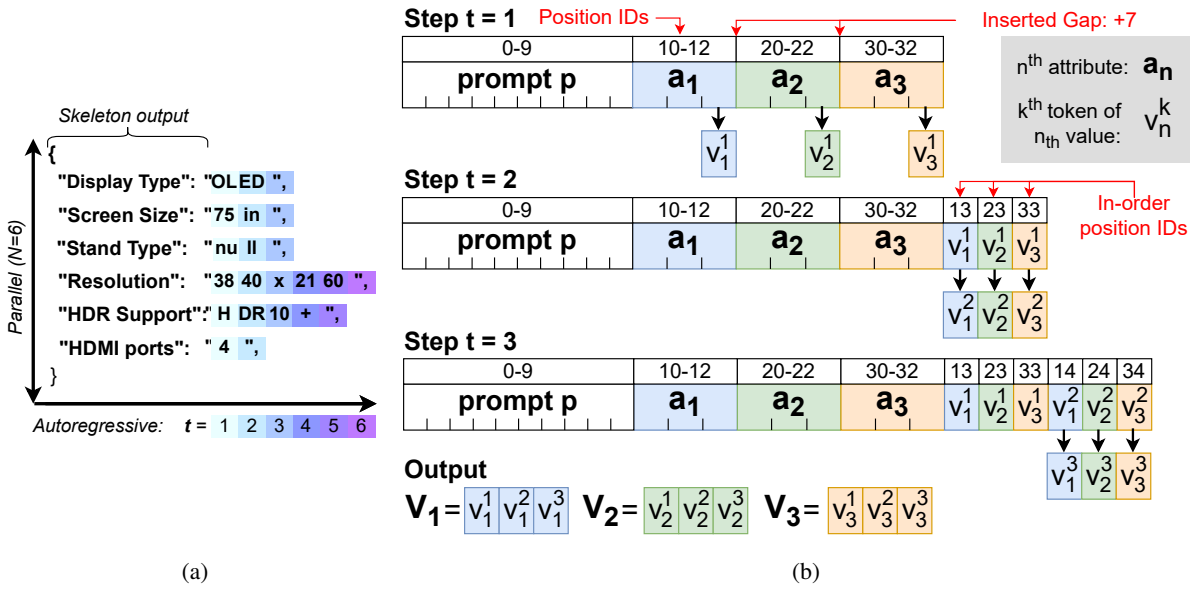
Specific implementation details are provided in Section 5. Our overarching objective is to maximize the throughput and minimize the cost, while maintaining state-of-the-art output quality.

4 Methodology

We first describe the standard autoregressive decoding process from a probabilistic view in Section 4.1, and then describe our modifications made for HPD in Section 4.2. We follow with a detailed breakdown of the HPD algorithm in Section 4.3.

4.1 Autoregressive Generation

In general LLM decoding, the Transformer architecture requires output tokens to be generated one at a time, i.e. *autoregressively*. This constraint is



(a) (b)

Figure 2: (a) An illustration of the generated output with HPD. The skeleton output defines the attributes to extract. The tokens within each value are generated autoregressively from left to right, while independent values are generated in parallel. (b) A block diagram for the HPD process: At step $t = 1$, the first token of each value is decoded in parallel using the next token prediction at each attribute end position. The position IDs shown above each token have an inserted gap between attributes, which is gradually filled by the generated value tokens at each step. New tokens are always appended to the sequence in memory which places them out of logical order, but the position IDs define the attention mask and positional embeddings during attention. In this example, three values of three tokens are decoded in parallel using only three inference steps. An alternative version is presented in Figure 7.

imposed by the attention blocks. At each step t, the current hidden state representation interacts with the attention keys and attention values derived from all previous token hidden states in the sequence. Since this hidden state is ultimately used to sample Y_t, there is a dependency for each Y_t on all $Y_{<t}$.

For AVE, let us view the output $\boldsymbol{y}$ in the the attribute/value pair representation, $\boldsymbol{y} = \{(\boldsymbol{a}_1, \boldsymbol{v}_1), (\boldsymbol{a}_2, \boldsymbol{v}_2), \ldots, (\boldsymbol{a}_N, \boldsymbol{v}_N)\}$. Let us denote $\boldsymbol{A}_n$ and $\boldsymbol{V_n}$ as the multivariate random variables representing the joint token distribution of the n-th attribute and value respectively. We can then express the full output random variable as a concatenation of the attribute and value random variables

$$\begin{aligned} \boldsymbol{Y}_n &= concat(\boldsymbol{A}_n, \boldsymbol{V}_n) \\ \boldsymbol{Y}_{1:N} &= concat(\boldsymbol{Y}_1, \boldsymbol{Y}_2, \ldots, \boldsymbol{Y}_N). \end{aligned} \quad (2)$$

Since the attributes $\boldsymbol{a}_n$ are known and provided in the prompt, we assume that their according random variable collapses to $P(\boldsymbol{A}_n = \boldsymbol{a}_n) = 1$ for the correct attribute. We therefore omit $\boldsymbol{A}_n$ from subsequent equations and replace it directly with $\boldsymbol{a}_n$. Since $\boldsymbol{y}$ is generated autoregressively, there exists a dependence between values:

$$\begin{aligned} P(\boldsymbol{Y}_{1:N} \mid \boldsymbol{p}) &= \prod_{n=1}^{N} P(\boldsymbol{V}_n \mid \boldsymbol{p}, \boldsymbol{a}_{1:n}, \boldsymbol{V}_{<n}). \quad (3) \\ &= \prod_{n=1}^{N} P(\boldsymbol{V}_n \mid \boldsymbol{p}, \boldsymbol{V}_{<n}). \end{aligned}$$

The dependence on the attributes $\boldsymbol{a}_{1:n}$ is omitted as it is already included in the prompt $\boldsymbol{p}$. Equation 3 shows that the n^{th} value is dependent through attention on the previous $n-1$ values.

4.2 Breaking the Autoregressive Chain

For the AVE task, we argue that this value dependence is a negative consequence of autoregressive generation. The task is to extract values directly from the document presented in the prompt $\boldsymbol{p}$. In theory, the extracted value for one attribute should not bias subsequent values. However, Shinzato et al. (2023) find that the ordering of attributes in the prompt can affect performance, as errors made in extracting the earlier values can cascade into errors and hallucinations in subsequent ones.

We propose eliminating the inter-value dependence by breaking the fundamental inter-token dependence that underpins autoregressive generation.

We assume **conditional independence** of generated values given the prompt $\boldsymbol{p}$:

$$P(\boldsymbol{Y}_{1:N} \mid \boldsymbol{p}) = \prod_{n=1}^{N} P(\boldsymbol{V}_n \mid \boldsymbol{p}). \tag{4}$$

Equation 4 describes our assumption that the probability of generating value $\boldsymbol{v}_n$ should not depend on previously generated values $\boldsymbol{v}_{<n}$. By breaking the chain dependence, the predicted probability distribution of each predicted value

$$\hat{P}(\boldsymbol{V}_n = \boldsymbol{v}_{n,1:K_n} \mid \boldsymbol{p}) = \prod_{k=1}^{K_n} \sigma(F(\boldsymbol{v}_{n,<k}, \boldsymbol{p})) \tag{5}$$

is independent from other values, although the K_n tokens that the value $\boldsymbol{v}_n$ comprises of must still be autoregressively generated. We can therefore compute $\hat{P}(\boldsymbol{V}_n = \boldsymbol{v})$ in parallel for all values, increasing the parallelism of the system. As a result, the number of LLM steps required for inference reduces from T to $max(K_n) \equiv K_{max}$.

By breaking the autoregressive generation chain into N smaller parallel chains for each value, we can express the predicted PDF of our output as:

$$\begin{aligned} \hat{P}(\boldsymbol{Y}_{1:N} = \boldsymbol{y} \mid \boldsymbol{p}) &= \prod_{n=1}^{N} \hat{P}(\boldsymbol{V}_n \mid \boldsymbol{p}) \\ &= \underbrace{\prod_{n=1}^{N}}_{parallel} \underbrace{\prod_{k=1}^{K_n} \sigma(F(\boldsymbol{v}_{n,<k}, \boldsymbol{p}))}_{autoregressive}. \end{aligned} \tag{6}$$

Next, we describe the algorithmic implementation of this process inside an LLM.

4.3 Hyper-Parallel Decoding (HPD)

4.3.1 Input and Position ID Construction

Given the assumed conditional independence of values in theory, we wish to parallelize the decoding of values using a standard decoder-only LLM. Figure 2 illustrates the basic Hyper-Parallel Decoding process, which we refer to throughout this section. We begin by constructing the model input $\boldsymbol{s}$ by concatenating the prompt $\boldsymbol{p}(\boldsymbol{x}, \boldsymbol{a}_{1:N})$ with a skeleton output template of the desired output. An example skeleton output is shown as the uncolored text in Figure 2 (a). In this template, we provide the attributes we wish to extract as keys, and leave the value field blank for the model to complete. For ease of notation, we simplify the output template to $\boldsymbol{s} = cat(\boldsymbol{p}, \boldsymbol{a}_{1:N})$, ignoring the structure tokens like tabs, "{" and "}". A block representation of $\boldsymbol{s}$ is illustrated in the first row of Figure 2 (b). In order to insert the value tokens into their respective positions during generation, a gap must be created in the output template. We introduce this gap by **skipping position IDs**. Position IDs are a variable used to define the absolute position of each token in a sequence, for example $\{0, 1, 2, 3, \dots\}$. LLMs are completely dependent on position IDs to encode the ordering of the input tokens, as they are used to define positional embeddings. At each position in $\boldsymbol{s}$ where a value is to be generated, we increment the position IDs by K_{max}, thus leaving a gap in the position ID sequence. This hyperparameter defines the maximum length of values that can be generated. We provide the exact position ID calculation algorithm in Appendix A.1. In Figure 2 (b), we insert a gap of $K_{max} = 7$ in the position IDs shown above each token block. This manipulation allocates space for the values to be generated without using any additional memory.

4.3.2 First Inference Step

During step $t = 1$, the LLM outputs the next-token probability distribution $\sigma(F(\boldsymbol{s}))$ for every token present in the input. Typically, only the last prediction is used, as it corresponds to the token probability for the next unknown token. We instead select the N next-token probabilities at the position where the missing values are, and sample N new tokens. These tokens make up the first token for each value $\{v_{1,1}, v_{2,1}, \dots, v_{N,1}\}$. Figure 2 (b) shows $N = 3$ tokens decoded from positions 12, 22 and 32. Because the ordering of tokens is defined by their position IDs, we can append the new tokens to the end of the sequence, and assign them the position IDs that would place them in the gaps created in Section 4.3.1. Appending new tokens to the end of the sequence is important for maintaining key-value cache functionality, since it is stored in a contiguous block of memory and cannot have values inserted mid-cache. In step $t = 2$ of Figure 2 (b), the first value tokens are given position IDs 13, 23 and 33, which would place them right after each of their respective attributes when ordering by position IDs. To maintain causal attention masking, the triangular attention mask is computed using the *position ID ordering* of the tokens rather than their in-memory ordering. Using scaled dot-product attention, the use of custom attention masks is supported. Figure 3 (a) illustrates how

the attention mask is modified with HPD, using single tokens to represent the prompt and attributes. The resulting mask is non-triangular but enforces causal attention in the position ID ordering.

Although the *ordering* is maintained, it is important to note that the values of the positions IDs are not, due to the existence of gaps in the sequence. We address this phenomenon later in Section 4.4.

4.3.3 Subsequent Inference Steps

In subsequent inference steps $t > 1$, only the N new tokens are passed to the model, along with the previously computed key-value cache. The LLM can therefore generate the t^{th} token for all N values. Figure 2 (a) illustrates the order of decoding: tokens are generated autoregressively from left to right, but in parallel along the y-axis. Position IDs are updated by incrementing the IDs used in the previous step, and the causal attention mask is computed with respect to these new positions.

4.3.4 Early Stopping

Since the length of individual values vary, we add a stop condition check at each inference step. For JSON decoding, we examine the generated token and prune values from the input sequence when the '\n' character is detected.[2] This pruning ensures that the key-value cache is not filled with irrelevant tokens. Generation ends either when every value is pruned or after K_{max} steps, where incomplete values are truncated.

4.3.5 Parallelism with Document Stacking

The current definition of the system generates at most N tokens in parallel per inference step, which is limited by the number of attributes we wish to extract. We propose adding an additional parallelism dimension by decoding the values from multiple documents within a single prompt: we construct $\boldsymbol{p}(\boldsymbol{x}_{1:J}, \boldsymbol{a}_{1:N})$ by stacking J documents in the same category. The instruction and attribute definition is efficiently shared among all documents, and the output of each document is stacked sequentially. Since the values of different products are also independent, we can now decode $J \times N$ tokens in parallel per inference step. Prior work has referred to this technique as *batch prompting* (Cheng et al., 2023), in the context of accelerating API calls. The speedup dynamics in this setting are very different, since the number of tokens decoded per step increases with J. Additionally, we do not ignore the associated performance penalty due to the longer prompt and larger key-value cache size, which reduces the maximum true batch size. In practice, we observe that significant gains can be attained from using $J > 1$, but that there exists a point of diminishing returns as we show in Figure 4. Appendix A illustrates the entire HPD algorithm with document stacking.

4.3.6 Support for Batch Inference

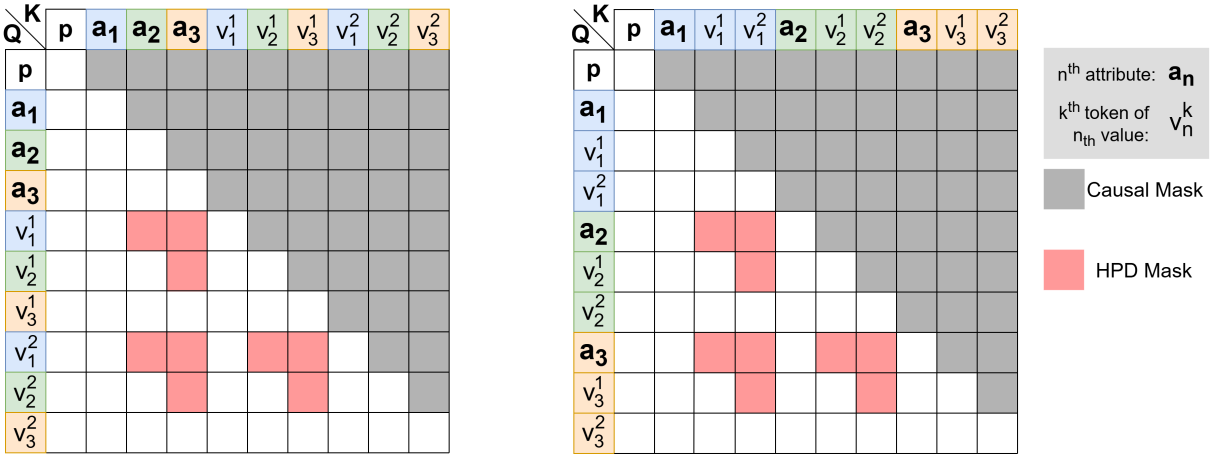
A crucial benefit of HPD is the support for batching multiple prompts through the LLM at once. All of the modifications above described happen within a single prompt, meaning that multiple prompts can still be processed in parallel with standard batched inference. When the number of incomplete values in each prompt becomes unbalanced, the length of the input tokens across batch elements is no longer equal, which is a requirement for combining batched inputs in tensor format. Therefore, the inputs are right padded with dummy tokens to maintain equal input lengths, and the dummy tokens are fully masked during attention. Combining all forms of parallelism, HPD can generate $b \times J \times N$ tokens per inference step for a batch size b, J stacked documents and N attributes to extract.

4.4 Fine-tuning and Distillation

As HPD makes no modification to the LLM architecture or model weights, it can be used directly on pre-trained LLMs. However, there exists a mismatch between the training data and test data. Firstly, the model can no longer attend to past values when generating a subsequent one, so the attention weights must be redistributed. Secondly, the use of K_{max} as a fixed position ID gap means that values shorter than K_{max} will cause some position IDs to be skipped, which affects the rotary positional embeddings during attention. We find that HPD can be used effectively with LLMs of several sizes at insignificant quality loss, and even quality gains in some scenarios (see Table 1).

To address the train/test mismatch and achieve equal performance to autoregressive decoding, a custom fine-tuning step for HPD is introduced. Given a set of in-domain ground truth labels, we format the training set to mimic the hyper-parallel decoding process by: (a) introducing the same position ID gaps as used in inference, and (b) modifying the causal attention mask to mask value tokens that would not have yet been generated during HPD. In particular, given a set of value labels

[2] Other structured formats require a different delimiter.



(a) Inference

(b) Fine-tuning

Figure 3: Illustration of the attention mask between queries Q (rows) and keys K (columns). Grey boxes indicate standard causal masking, while red boxes indicate the additional HPD mask applied. (a) During inference, decoded value tokens v_n^k are placed out of order in memory relative to their position IDs. The HPD mask ensures causality by blocking attention to attribute and value tokens that appear later in the position ID ordered sequence. (b) The LLM is fine-tuned using ground truth or pseudo ground truth labels in the regular token order. The HPD mask addresses the train/test mismatch by masking attention between tokens which cannot attend to each other during inference due to the parallel decoding process. The attention pattern between (a) and (b) is identical.

$\{\boldsymbol{v}_{1,1:K_1}, \boldsymbol{v}_{2,1:K_2}, \ldots, \boldsymbol{v}_{N,1:K_N}\}$, we apply a mask such that each token $v_{n,k}$ cannot attend to tokens $\{v_{n',k'} \, \forall \, n' < n, \, \forall \, k' > k\}$. Similarly, we prevent all attribute tokens from attending to all value tokens, since they will not be available in the first inference step. This mask is applied in addition to the standard causal mask, which prevents tokens from attending to those with a higher position ID. Figure 3 (b) illustrates the attention mask during fine-tuning. The modified mask replicates the attention pattern observed during inference (Figure 3 (a)): when decoding the k^{th} token of a value, the $k+1$ to K_n tokens of all values are missing and thus cannot be attended to.

When no ground truth labels are available, the fine-tuning set can be created as pseudo ground truth from a teacher LLM (Knowledge Distillation), or from the LLM itself using autoregressive generation (self-supervision). In AVE, this process can often be combined with the fine-tuning step that is already performed to align a base LLM to the task, therefore incurring zero additional training cost.

5 Experimental Setup

Datasets We evaluate HPD on two standard widely-used AVE benchmarks, OA-Mine (Zhang et al., 2022) and AE110k (Xu et al., 2019b). We select benchmarks containing human-annotated or verified labels over machine-labeled benchmarks such as MAVE that have been found to be more error-prone (Zou et al., 2024). We also introduce a third large-scale benchmark based on Amazon Reviews 2023 (Hou et al., 2024), which contains both product titles and descriptions. This benchmark is significantly larger in size, more challenging and more costly to extract attributes from. It requires a zero-shot setting as no labels are available. To address this, we employ knowledge distillation, fine-tuning the local LLMs on the zero-shot outputs of GPT-4.1. Table 2 describes the data split for each dataset, and Appendix B.1 provides the dataset details.

Models We use a set of state-of-the-art LLMs across a wide range of model sizes. GPT-4.1 is used as the most performant and expensive baseline[3]. Since the model is locked behind an API, HPD cannot be used. We instead employ a few-shot setting on OA-Mine and AE110k, which has been shown to provide strong performance (Brinkmann et al., 2024b). For more cost-efficient inference, we select the Qwen3 family of models in 1.7B, 4B, 8B and 32B sizes (Qwen Team, 2025) as a representative range of highly performant decoder-only LLMs. We also include Phi-4 14B (Microsoft Research, 2024) to represent other model families. Each model is fine-tuned with LoRA. Appendix B

[3]For OA-Mine and AE-110k, we report gpt-4-0613 results from Brinkmann et al. (2024b).

Model		OA-Mine		AE-110K		Amazon Reviews 2023			
						Base model		Fine-tuned model	
		F1	$/1k P.	F1	$/1k P.	$F1_{LLM}$	$/1k P.	$F1_{LLM}$	$/1k P.
GPT-4(.1) (0-shot)		0.681	N/A	0.621	N/A	**0.871**	1.14	N/A	N/A
GPT-4(.1) (10-shot)		0.822	32.15	**0.875**	17.85	N/A	N/A	N/A	N/A
Qwen3-32B	AR	0.888	0.688	0.794	0.727	0.791	5.347	**0.884**	7.745
	HPD	0.878	0.267	0.854	0.270	0.797	0.805	**0.884**	0.898
Phi4-14B	AR	0.880	0.355	0.797	0.350	0.839	2.675	0.876	3.420
	HPD	0.888	0.095	0.782	0.109	0.814	0.216	0.883	0.248
Qwen3-8B	AR	**0.891**	0.166	0.787	0.163	0.646	1.142	0.874	1.362
	HPD	0.878	0.069	0.853	0.070	0.751	0.132	0.881	0.167
Qwen3-4B	AR	0.878	0.147	0.795	0.156	0.671	1.120	0.861	1.200
	HPD	0.876	0.061	0.839	0.063	0.732	0.127	0.870	0.133
Qwen3-1.7B	AR	0.863	0.080	0.780	0.076	0.510	0.887	0.861	1.110
	HPD	0.849	0.034	0.814	0.035	0.529	0.090	0.870	0.095

Table 1: F1 scores and cost per 1k products of selected models for AVE on our selected datasets between the autoregressive (AR) baseline and our proposed HPD. Bold numbers indicate the highest F1 score per dataset. LLM-as-a-judge F1 scores are used for Amazon Reviews.

Dataset	Train	Test	Attr./Cat.
OA-Mine	1,452	491	10.63
AE110k	1,568	524	10.10
Amazon Reviews	45,000	17,905	16.00

Table 2: The data split breakdown. Attr./Cat. shows the average number of attributes per product category. For Amazon Reviews, pseudo-labels are generated using GPT-4.1 to form the training data.

contains the full inference and fine-tuning details.

Metrics We report cost and throughput as defined in Section 3. We report the F1 score achieved by each model on OA-Mine and AE110k. For Amazon Reviews, Claude 3.5 Sonnet (Anthropic, 2024) is used as an unbiased judge to evaluate the correctness of extracted values. We report an F1 score derived from the LLM evaluation. The prompt and calculation details are provided in Appendix E.

6 Results

Table 1 reports the performance and cost of each model with autoregressive decoding (AR) and Hyper-Parallel Decoding (HPD), on all datasets. Results show that HPD reduces the cost of inference by up to **3.73X**, **3.21X**, and **13.79X** on OA-Mine, AE110K and Amazon Reviews for Phi4-14B. The cost reduction is directly proportional to the increase in throughput, given in Appendix C for Amazon Reviews. For the fine-tuned LLMs, this cost reduction comes with no quality penalty:

Amazon Reviews 2023		
Model	Prod./s	Speedup
Qwen3-32B HPD	0.539	10.78 X
Qwen3-32B + 1.7B Spec.	0.104	2.08 X
Qwen3-32B + 4B Spec.	0.092	1.84 X
Qwen3-32B + 8B Spec.	0.063	1.26 X
Qwen3-32B AR	0.050	1.00 X

Table 3: Inference throughput in product/s of HPD compared to speculative decoding on Amazon Reviews. Qwen3-32B is used as the base model, with differently-sized draft models of the same family. The second column shows speedup relative to standard autoregressive (AR) decoding. Results are shown with no batching and no document stacking.

The average F1 score for HPD is on average 0.7% lower for OA-Mine, but 4.5% higher for AE110k and 0.7% higher for Amazon reviews. We make four key observations:

HPD's performance gains depend on the task, but are consistent across a range of model sizes. The obtained cost reduction on Amazon Reviews is significantly higher than the other datasets. The document context size and number of attributes per category play a significant role in the performance gains of HPD. HPD can parallelize more effectively when there are more attributes to extract, and longer documents reduce the maximum batch size that autoregressive inference can use. Given a fixed

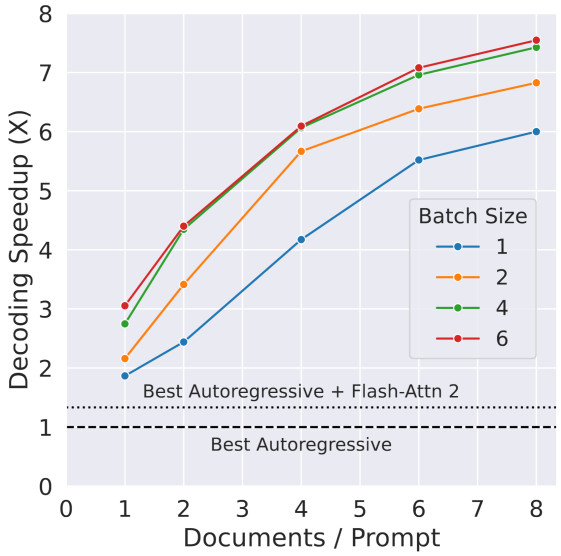


Figure 4: Throughput (products / s) increase from HPD relative to autoregressive decoding on Qwen3-8B for varying documents per prompt and batch size on Amazon Reviews 2023. HPD achieves up to 7.55X speedup. The autoregressive baseline maximizes batch size given VRAM constraints, shown with standard SPDA and Flash-Attention 2.

dataset, the speedup from HPD is mostly consistent across model sizes ranging from 1.7B to 32B.

HPD is plug-and-play. When using the base models for Amazon Reviews, we observe that the models using HPD perform on par or better than their autoregressive counterparts on 5/6 of the models tested. This demonstrates that HPD is not limited by requiring fine-tuning.

HPD is preferable over speculative decoding. We compare HPD directly to speculative decoding on the Amazon Reviews dataset. We select Qwen3-32B as the base model and Qwen3-8B,4B and 1.7B as the draft models, using the Hugging Face Transformers dynamic thresholding implementation. Batching is disabled for this experiment because it is not supported by speculative decoding. Table 3 shows that speculative decoding can increase throughput by up to 2.08X compared to autoregressive decoding on this task. HPD significantly outperforms speculative decoding by achieving a 10.78X speedup. The speedup difference can be attributed to the fact that HPD completely skips the draft phase and can still parallelize inference of the base model, while maintaining output quality.

HPD can linearly scale inference throughput with document stacking. Figure 4 illustrates how the relative throughput of Qwen3-8B HPD improves with regards to batch size and number of stacked documents. The autoregressive baseline uses the maximum allowable batch size under VRAM constraints. We observe that throughput scales better with document stacking than batch size. The added parallelism from document stacking linearly increases throughput up to a saturation point, where added parallelism becomes ineffective. Extrapolating our throughput gains to a corpus of 500M Amazon Reviews-like documents using the cost-effective Qwen3-8B, HPD would achieve a cost saving of $597,000 over autoregressive decoding, and $486,000 against GPT-4.1.

Human evaluation results are aligned with the LLM judge preference on Amazon Reviews. To validate the LLM judge used for evaluating Amazon Reviews, we conduct a human evaluation experiment on a subset of our results in Appendix D, showing agreement between the human annotators and the LLM judge. In a blind test, human annotators slightly prefer Qwen3-8B HPD over its autoregressive counterpart as well as GPT-4.1, which aligns with the performance ordering according the the LLM judge.

7 Conclusion

Large Language Models excel at solving a wide range of problems, but the sequential nature of autoregressive generation imposes a bottleneck on performance. In some tasks such as AVE, the generated outputs can be broken down into independent components. Hyper-Parallel Decoding leverages this independence to enable LLMs to generate multiple tokens in parallel, without requiring any architectural or model weight modifications. HPD parallelizes generation within each prompt, sharing both memory and computation while synergizing with batched inference, crucial for the offline setting. By manipulating the input prompt, position IDs and attention mask, HPD maintains functionality with key existing methods such as key-value caching, demonstrating a zero-compromise throughput increase and cost reduction on a range of datasets in the product AVE domain. HPD can in theory generalize beyond AVE to other tasks with similar independent structures, which we aim to explore in future work.

Limitations

Models We perform our experiments primarily using the Qwen3 model family, as well as Phi4. Experiments using other model families and LLMs exceeding 32B parameters are omitted due to computational constraints.

Datasets In this work, we evaluated the performance of Hyper-Parallel Decoding exclusively on product AVE, although the method does not require product data. We leave for future work to demonstrate the applicability of HPD in other domains and other tasks beyond AVE.

Compatibility Hyper-Parallel Decoding accelerates inference by reducing the total number of inference steps during generation, and is orthogonal to most other acceleration methods. We have shown compatibility with quantization and knowledge distillation, but have not shown how other methods such as paged attention and Mixture-of-Experts interact with HPD. One incompatibility of note is Flash-Attention (Dao, 2023), which assumes a triangular attention mask in its current implementation. However, new implementations such as FlexAttention (Dong et al., 2024) now provide similar benefits to FlashAttention with the flexibility of custom attention masks. We provide an implementation of HPD with both FlexAttention and standard scaled dot-product attention (SDPA), although the latter is used in our experiments.

Evaluation We observed that the labels for AE110k and OA-Mine do not contain all present values, sometimes leading to the LLM being penalized for extracting a correct attribute. For Amazon Reviews 2023, evaluation is performed by an LLM judge due to resource constraints. Although our human evaluation agreed with the LLM judge on a small subset of data, it is not a perfect substitute.

References

Monica Agrawal, Stefan Hegselmann, Hunter Lang, Yoon Kim, and David Sontag. 2022. Large language models are few-shot clinical information extractors. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1998–2022. Association for Computational Linguistics.

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 268–284. Association for Computational Linguistics.

Anthropic. 2024. Claude 3.5 sonnet. https://www.anthropic.com.

Anthropic. 2025. Claude 3.7 sonnet. https://www.anthropic.com.

Ansel Blume, Nasser Zalmout, Heng Ji, and Xian Li. 2023. Generative models for product attribute extraction. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 575–585, Singapore. Association for Computational Linguistics.

Alexander Brinkmann, Nick Baumann, and Christian Bizer. 2024a. Using llms for the extraction and normalization of product attribute values. In *Advances in Databases and Information Systems: 28th European Conference, ADBIS 2024, Bayonne, France, August 28–31, 2024, Proceedings*, page 217–230. Springer-Verlag.

Alexander Brinkmann, Roee Shraga, and Christian Bizer. 2024b. Extractgpt: Exploring the potential of large language models for product attribute value extraction. In *Information Integration and Web Intelligence: 26th International Conference, IiWAS 2024, Bratislava, Slovak Republic, December 2–4, 2024, Proceedings, Part I*, page 38–52. Springer-Verlag.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. Batch prompting: Efficient inference with large language model APIs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 792–810. Association for Computational Linguistics.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. ArXiv preprint: arXiv 2307.08691.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. 2024. Flex attention: A programming model for generating optimized attention kernels. ArXiv preprint: arXiv 2412.05496.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of llm inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Yupeng Hou, Jiacheng Li, Zhankui He, An Yan, Xiusi Chen, and Julian McAuley. 2024. Bridging language and items for retrieval and recommendation. *arXiv preprint arXiv:2403.03952*.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. ArXiv preprint: arXiv 1508.01991.

Hyunwoo Hwangbo, Yang Sok Kim, and Kyung Jin Cha. 2018. Recommendation system development for fashion retail e-commerce. *Electronic Commerce Research and Applications*, 28:94–101.

Michal Kvet, Miroslav Potočár, and Slavomír Tatarka. 2025. Real estate attribute value extraction using large language models. *IEEE Access*, 13:73076–73095.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proc. Int. Conf. Mach. Learn. (ICML)*.

Feng Lin, Hanling Yi, Yifan Yang, Hongbin Li, Xiaotian Yu, Guangming Lu, and Rong Xiao. 2025. Bita: Bi-directional tuning for lossless acceleration in large language models. *Expert Systems with Applications*, 279:127305.

Mingdao Liu, Aohan Zeng, Bowen Wang, Peng Zhang, Jie Tang, and Yuxiao Dong. 2024. Apar: Llms can do auto-parallel auto-regressive decoding. ArXiv preprint: arXiv 2401.06761.

Hanqing Lu, Youna Hu, Tong Zhao, Tony Wu, Yiwei Song, and Bing Yin. 2021. Graph-based multilingual product retrieval in E-commerce search. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, pages 146–153, Online. Association for Computational Linguistics.

Microsoft Research. 2024. Phi-4 technical report. ArXiv preprint: arXiv 2412.08905.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. 2024. Skeleton-of-thought: Prompting llms for efficient parallel generation. ArXiv preprint: arXiv 2307.15337.

OpenAI. 2025. Gpt-4.1. https://openai.com/index/gpt-4-1/.

Qwen Team. 2025. Qwen3 technical report. ArXiv preprint: arXiv 2505.09388.

Kalyani Roy, Pawan Goyal, and Manish Pandey. 2024. Exploring generative frameworks for product attribute value extraction. *Expert Systems with Applications*, 243:122850.

Ohad Rozen, David Carmel, Avihai Mejer, Vitaly Mirkis, and Yftah Ziser. 2021. Answering product-questions by utilizing questions from other contextually similar products. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 242–253, Online. Association for Computational Linguistics.

Kassem Sabeh, Robert Litschko, Mouna Kacimi, Barbara Plank, and Johann Gamper. 2024. An empirical comparison of generative approaches for product attribute-value identification. ArXiv preprint: arXiv 2407.01137.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. ArXiv preprint: arXiv 1911.02150.

Keiji Shinzato, Naoki Yoshinaga, Yandi Xia, and Wei-Te Chen. 2023. A unified generative approach to product attribute-value identification. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 6599–6612, Toronto, Canada. Association for Computational Linguistics.

Quoc-Tuan Truong, Tong Zhao, Changhe Yuan, Jin Li, Jim Chan, Soo-Min Pantel, and Hady W. Lauw. 2022. Ampsum: Adaptive multiple-product summarization towards improving recommendation captions. In *Proceedings of the ACM Web Conference 2022*, WWW '22, page 2978–2988, New York, NY, USA. Association for Computing Machinery.

Damir Vandic, Jan-Willem van Dam, and Flavius Frasincar. 2012. Faceted product search powered by the semantic web. *Decision Support Systems*, 53(3):425–437.

Qifan Wang, Li Yang, Bhargav Kanagal, Sumit Sanghai, D. Sivakumar, Bin Shu, Zac Yu, and Jon Elsas. 2020. Learning to extract attribute value from product via question answering: A multi-task approach. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 47–55. Association for Computing Machinery.

Liqiang Xiao, Jun Ma, Xin Luna Dong, Pascual Martínez-Gómez, Nasser Zalmout, Chenwei Zhang, Tong Zhao, Hao He, and Yaohui Jin. 2021. End-to-end conversational search for online shopping with utterance transfer. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3477–3486, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Huimin Xu, Wenting Wang, Xin Mao, Xinyu Jiang, and Man Lan. 2019a. Scaling up open tagging from tens to thousands: Comprehension empowered attribute value extraction from product title. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5214–5223. Association for Computational Linguistics.

Huimin Xu, Wenting Wang, Xin Mao, Xinyu Jiang, and Man Lan. 2019b. Scaling up open tagging from tens to thousands: Comprehension empowered attribute value extraction from product title. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5214–5223, Florence, Italy. Association for Computational Linguistics.

Li Yang, Qifan Wang, Zac Yu, Anand Kulkarni, Sumit Sanghai, Bin Shu, Jon Elsas, and Bhargav Kanagal. 2022. Mave: A product dataset for multi-source attribute value extraction. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, page 1256–1265. Association for Computing Machinery.

Liyi Zhang, Mingzhu Zhu, and Huang Wei. 2009. A framework for an ontology-based e-commerce product information retrieval system. *Journal of Computers*, 4.

Wenxuan Zhang, Yang Deng, Jing Ma, and Wai Lam. 2020. AnswerFact: Fact checking in product question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2407–2417, Online. Association for Computational Linguistics.

Xinyang Zhang, Chenwei Zhang, Xian Li, Xin Luna Dong, Jingbo Shang, Christos Faloutsos, and Jiawei Han. 2022. Oa-mine: Open-world attribute mining for e-commerce products with weak supervision. In *Proceedings of the ACM Web Conference 2022*, WWW '22, page 3153–3161. Association for Computing Machinery.

Zikang Zhang, Wangjie You, Tianci Wu, Xinrui Wang, Juntao Li, and Min Zhang. 2025. A survey of generative information extraction. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 4840–4870, Abu Dhabi, UAE. Association for Computational Linguistics.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. 2024. A survey on efficient inference for large language models. ArXiv preprint: arXiv 2404.14294.

Henry Peng Zou, Vinay Samuel, Yue Zhou, Weizhi Zhang, Liancheng Fang, Zihe Song, Philip S. Yu, and Cornelia Caragea. 2024. ImplicitAVE: An open-source dataset and multimodal LLMs benchmark for implicit attribute value extraction. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 338–354, Bangkok, Thailand. Association for Computational Linguistics.

Appendix

A The HPD Algorithm

Algorithm 1 describes the entire Hyper-Parallel Decoding algorithm in pseudo code, leaving out the manipulation of position IDs, which is described in Appendix A.1. Figure 7 contains is an alternative illustration to Figure 2.

Algorithm 1 Hyper-Parallel Decoding

$\boldsymbol{p} \leftarrow p(\boldsymbol{x}_{1:J}, \boldsymbol{a}_{1:N})$
$\boldsymbol{s} \leftarrow cat(\boldsymbol{p}, \boldsymbol{a}_{1:N})$
$\boldsymbol{values} = empty(J, N, K_{max})$
$t \leftarrow 0$
while $t < K_{max}$ and not all $\boldsymbol{values}$ pruned **do**
  if $t = 0$ **then**
    $\boldsymbol{d}, \boldsymbol{c} \leftarrow \sigma(F(\boldsymbol{s}))$ ▷ Prefill
  else
    $\boldsymbol{d}, \boldsymbol{c} \leftarrow \sigma(F(\boldsymbol{d}, \boldsymbol{c}))$ ▷ Autoregressive
  end if
  $\boldsymbol{d} \leftarrow prune(\boldsymbol{d})$
  for $j \in range(J)$ **do**
    for $i \in range(N)$ **do**
      if $\boldsymbol{values}[j, i]$ not pruned **then**
        $\boldsymbol{values}[j, i, t] \leftarrow \boldsymbol{d}[jN + i]$
      end if
    end for
  end for
  $k \leftarrow k + 1$
end while

Figure 5: The basic Hyper-Parallel Decoding algorithm. Given a prompt $\boldsymbol{p}$ to extract N values for attributes $\boldsymbol{a}_{1:N}$ from J products $\boldsymbol{x}_{1:J}$, perform up to K_{max} inference steps, where K_{max} is the maximum value length. Each step generates $j \times N$ tokens $\boldsymbol{d}$, and updates the key-value cache $\boldsymbol{c}$. Step t generates the t^{th} token off all values in parallel, and completed values are pruned. The indexing used assumes no pruning for illustrative purposes.

A.1 Position ID calculation

We begin by assigning the input $\boldsymbol{s} = cat(\boldsymbol{p}, \boldsymbol{a}_{1:N})$ containing the prompt and skeleton output with standard initial position IDs $pos_{id} = [0, 1, 2, \ldots, |s|]$. Let start($\boldsymbol{a}_i$), end($\boldsymbol{a}_i$) be the start and end token index of attribute $\boldsymbol{a}_i$ in $\boldsymbol{s}$. We then insert position ID spacing of K_{max} between each attribute according to Algorithm 2. During inference, the k^{th} token of the n^{th} attribute $v_{n,k}$ is assigned position ID: $pos_{id}[\text{end}(\boldsymbol{a}_n)] + k$.

Algorithm 2 Position ID assignment

$\boldsymbol{s} \leftarrow cat(\boldsymbol{p}, \boldsymbol{a}_{1:N})$
$pos_{id} = range(|\boldsymbol{s}|)$
offset $\leftarrow 0$
for $i \in range(N)$ **do**
  $pos_{id}[\text{start}(\boldsymbol{a}_i)\text{:end}(\boldsymbol{a}_i) + 1] \mathrel{+}= \text{offset}$
  offset $\leftarrow$ offset $+ K_{max}$
end for

Figure 6: Position ID assignment algorithm for the model input $\boldsymbol{s}$. After assigning initial position IDs, a space of K_{max} is inserted in between each attribute for the decoded values to be inserted.

B Additional Experimental Details

B.1 Dataset Details

OA-Mine (Zhang et al., 2022): We use the human-annotated subset of OA-Mine, containing 9,811 attribute-value pairs for 1,943 products across 10 product types from Amazon.com. The data consists of only product titles. We follow the same (large) train/test split used by (Brinkmann et al., 2024b).

AE110k (Xu et al., 2019b) AE110k contains 39,505 product titles from AliExpress Sports & Entertainment, with the label values obtained directly from their structured product catalog. There are a total of 2,045 unique attributes and 10,977 unique values. We follow the same (large) train/test split used by (Brinkmann et al., 2024b).

Although these two AVE datasets contain high quality labels useful for correctness evaluation, they differ substantially from the large-scale e-commerce datasets where LLM inference cost is a primary concern. Firstly, product information is not only contained in product titles, but also in product descriptions. The inclusion of product description not only makes the task more challenging, but also more costly due to the longer product context and number of attributes present. Secondly, products from large e-commerce services have a wider range of product categories than the subsets selected in OA-Mine and AE110k. Therefore, acquiring ground truth labels is often costly or infeasible, requiring a zero-shot setting. We therefore craft a new benchmark using the open source product data from Amazon Reviews 2023 (Hou et al., 2024):

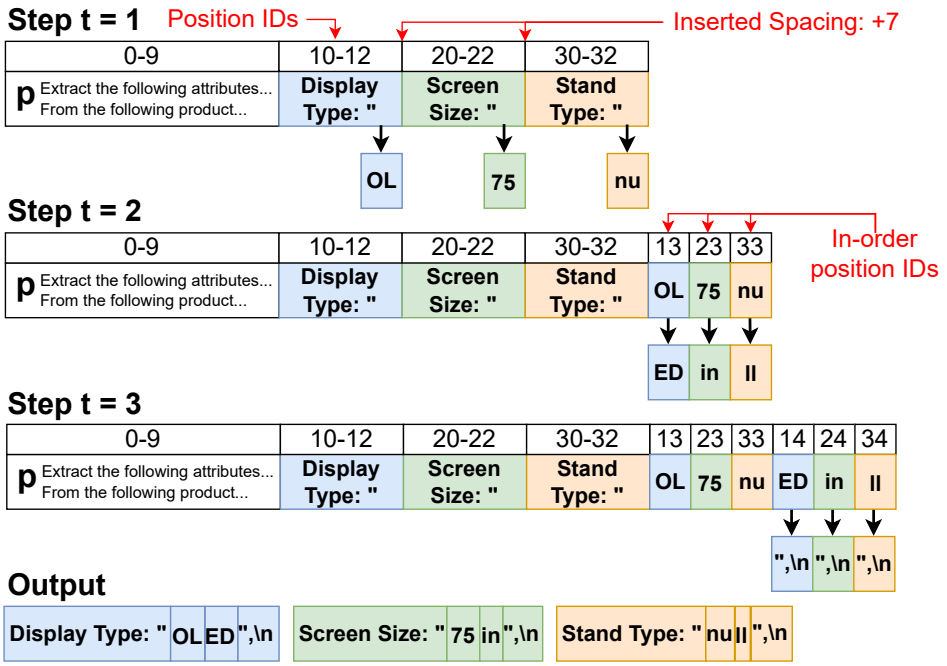


Figure 7: Alternative block diagram for the HPD process: At step $t = 1$, the first token of each value is decoded in parallel using the next token prediction at each attribute end position. The position IDs shown above each token have an inserted gap between attributes, which is gradually filled by the generated value tokens at each step. New tokens are always appended to the sequence in memory which places them out of logical order, but the position IDs define the attention mask and positional embeddings during attention. In this example, three values of three tokens each are decoded in parallel using only three inference steps.

Amazon Reviews: We collect 900k products across 31 product categories. We use Claude 3.7 Sonnet (Anthropic, 2025) to define the 16 most important attributes for each product category, resulting in 267 unique attributes in total. Due to the absence of labeled training data, we generate zero-shot predictions using GPT-4.1 with batched API (OpenAI, 2025) on 45k uniformly sampled products, using their titles, descriptions, and bullet points as input. We then use these predictions as pseudo-labels to fine-tune small language models (SLMs) through knowledge distillation. Evaluation is conducted on a separate sample of 18k products. The relevant prompts are provided in Appendix E.

B.2 Fine-tuning and Inference Details

All experiments are run on the Hugging Face Transformers framework with Accelerate. We fine-tune the local models for 5 epochs on OA-Mine and AE110k. A separate fine-tuning is performed for standard autoregressive inference and HPD using the custom alignment process described in Section 4.4. DeepSpeed Zero3 and LoRA are used to efficiently fine-tune even the Qwen3-32B model. On Amazon Reviews, we apply knowledge distillation for 1 epoch. The learning rate is tuned as a hyperparameter for the lowest validation loss. For Qwen3-4B and 1.7B, we use full-parameter fine-tuning.

At inference, we stack 6 documents per prompts, and use a maximum value length $K_{max} = 30$. 4-bit quantization is used for Qwen3-32B with NF4 using BitsandBytes. For smaller models, we use bf16 inference as we find quantization to not improve inference speed given the available VRAM. The batch size for each model is adjusted to max out the VRAM utilization and ensure fair comparison between the larger stacked prompts in HPD and standard autoregressive inference. We use Flash-Attention-2 for autoregressive inference, but revert to SPDA for HPD since Flash-Attention does not support modified attention masks by default.

B.3 Resources

We select the highly performant Amazon EC2 g6e.48xlarge server for local fine-tuning and inference, using 8-way data parallelism on Nvidia L40S 48GB GPUs. The cost/product of the local

Model		Amazon Reviews 2023 LLM-F1	Prod./s
Qwen3-32B	AR	0.884	0.14
	HPD	0.884	1.17
Phi4-14B	AR	0.876	0.31
	HPD	0.883	4.23
Qwen3-8B	AR	0.874	0.77
	HPD	0.881	6.28
Qwen3-4B	AR	0.861	0.87
	HPD	0.870	7.89
Qwen3-1.7B	AR	0.861	0.94
	HPD	0.870	10.99

Table 4: LLM-as-a-judge F1 scores and throughput (products/s) of selected fine-tuned local models for Amazon Reviews. We compare the autoregressive (AR) and hyper-parallel (HPD) performance for each local model.

models is derived from the time required to process all products on the test set on this instance and the on-demand rental cost of $30.13/h as of July 2025. We select this instance because it is publicly available and representative of the type of server that would be used for efficiently processing millions of products for AVE.For API based LLMs, we define cost as the average API credit cost/product ($/1k products) as of July 2025. This cost already includes a discount for prefix caching.

C Throughput Measurements on Amazon Reviews

Table 4 shows the throughput in products processed per second for the local fine-tuned models on Amazon Reviews. We observe a relatively constant throughput increase of 10X across model sizes.

D Human Evaluation on Amazon Reviews 2023

To further validate the LLM-as-a-judge F1 scores on Amazon Review 2023, we conducted a blind human evaluation on 563 examples to assess the quality of LLM evaluation. Four expert annotators performed pairwise comparisons between outputs from GPT-4.1, Qwen-8B (AR), and Qwen3-8B (HPD), deciding which model's extractions are most faithful to the product context. For each comparison, annotators examined the original review text alongside attribute-value pairs from two randomly selected models, then judged whether one output was superior (win), inferior (loss), or comparable (tie) to the other.

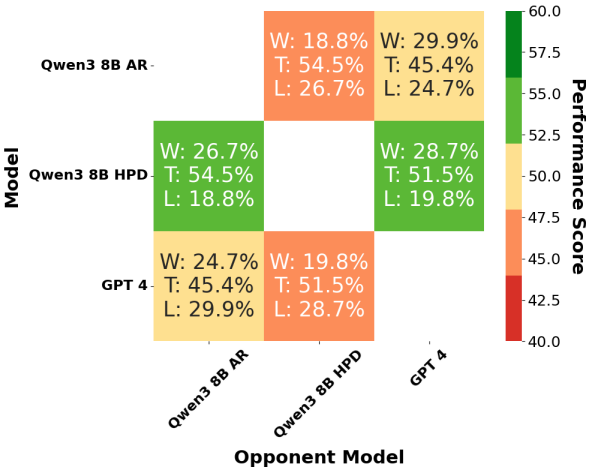


Figure 8: Amazon Reviews 2023 dataset: Matrix of win (W), tie (T) and loss (L) rate of GPT-4.1, standard Qwen3-8B (AR), and Qwen3-8B with HPD. Wins, ties and losses are determined by human annotators randomly comparing 2/3 model outputs for one product, and judging which set of values are most faithful to the product context. Performance score is calculated as (W+T)/(T+L) normalized to 0-100.

Figure 8 presents the comparison matrix with win/tie/loss rates across the three models. Qwen3-8B (HPD) achieves higher win rate against GPT-4.1 (28.7% vs. 19.8%), surpassing even the teacher model in human preference. When comparing the two student models directly, Qwen3-8B (HPD) also outperforms its AR counterpart—winning 26.7% of match-ups while losing only 18.8% and with 54.5% ties, demonstrating that our approach not only achieves comparable output quality against the teacher model and the autoregressive approach, but also drastically reduces the inference time.

E Example Prompts and Outputs

Figures 9, 10, 11 contain the prompts for extracting attribute values, defining the product category attributes and evaluating the quality of extracted values for Amazon Reviews. Using the defined classification categories defined in Figure 11, we define C = "correct", CN = "correct null", I = "incorrect", M = "missing" and H = "hallucination". The LLM F1 score is calculated as:

$$\text{LLM-P} = \frac{C + CN}{C + CN + H + I}$$

$$\text{LLM-R} = \frac{C + CN}{C + CN + M + I}$$

$$\text{LLM-F1} = \frac{2 \times \text{LLM-P} \times \text{LLM-R}}{\text{LLM-P} + \text{LLM-R}}$$

```
You are an expert at extracting product details.

Consider the following product attributes:

{attributes}

Analyze each input <product> element below and extract all of the attributes specified above in the
    requested standard format.

Output a JSON dict with a key for each input product ID, and a nested dict with a key for each
    attribute (use the attribute name) and the extracted value.
Escape any output double quotes characters (") to ensure a valid output JSON.

The values in the standard format:
    - must preserve names of characters, themes, brands, patterns, flavors if present.
    - for Product Type, Product Intent, Sustainability attributes: must preserve details like method
    names and types without aggregating.
    - must be 4-5 words or less.
    - must retain all the different numbers and their standard units (e.g., counts, packs), without
    multiplying them together.
    - must not use placeholders like '-' or 'null' for concise values.
    - must simplify words to base noun forms without plurals, uppercase, adjective or adverbial
    forms.
    - must write null if the value of an attribute is not explicitly decribed in the product
    information.

Put the JSON list in <result> tags.

Input products:

{products}
```

Figure 9: Amazon Reviews Prompts

```
You are an expert at comparing products to determine if they are equivalent in terms of function,
    specification, form, design, material, quantity, quality, brand value.

Analyze the <product> elements below, and output a list of price-sensitive attributes that
    experienced customers and product category experts would consider when comparing these products
    to see if they are exact equivalents, very similar, or incompatible.

For each attribute write a brief description which lists 3-6 typical values as examples.
Make sure each attribute name is not more than 4 words.
Do not include attributes related to price.
For each attribute output the data type of its value as numerical or categorical.

Provide up to 16 attributes, including fine-grained ones.

Only output a list of XML <attribute> elements with numeric id (as element attribute), product
    attribute <name>, <description> and <datatype>.

Products:

{products}
```

Figure 10: Attribute definition prompt for Claude 3.7 Sonnet

```
You are an expert at evaluating product details.

Consider the following product attributes:

{attributes}

An existing system has attempted to extract the value of each attribute from the product information
    below. Analyze the input context and extracted values to determine if the extracted values are
    correct.

Classify each attribute into 1 of 5 types:
- "correct": The input text contains sufficient information to extract the attribute value beyond
    resonable doubt. The attribute value extracted is correct and complete. Words that convey the
    same meaning are allowed.
- "incorrect": The input text contains sufficient information to extract the attribute value beyond
    resonable doubt. The attribute value extracted is incorrect or incomplete. Do no include "null"
    values in this category.
- "missing": The input text contains sufficient information to extract the attribute value beyond
    resonable doubt. The attribute value extracted is "null" or missing.
- "correct null": The value cannot be directly inferred from the input context. The attribute value
    extracted is "null" or similar. Values indicating the absense of an attribute ("no", "none")
    should be "correct null".
- "hallucination": The value cannot be directly inferred from the input context. An attribute value
    was extracted which is not indicating the absence of a value.

Use the following explanation structure for each attribute:
<attribute_name>: The value "<value>" is <classification> because <brief explanation>.

After the explanation, output a JSON dict with a key for each attribute (use the attribute name) and
    one of the evaluation results as value (correct, incorrect, missing, correct null,
    hallucination). Wrap the JSON around <result></result> tags. Do not provide explanations.

Input product:

{product}

Extracted Values:

{values}
```

Figure 11: Evaluation prompt used for Claude 3.5 Sonnet