# Encode Once and Decode in Parallel: Efficient Transformer Decoding

Anonymous ACL submission

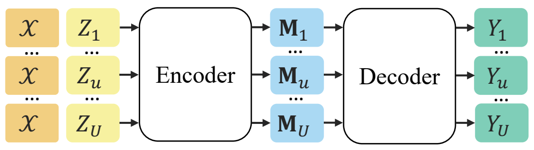
## Abstract

Transformer-based NLP models are powerful but have high computational costs that limit deployment scenarios. Finetuned encoder-decoder models are popular in specialized domains and can outperform larger more generalized decoder-only models, such as GPT-4. We introduce a new configuration for encoder-decoder models that improves efficiency on structured output and question-answering tasks where multiple outputs are required of a single input. Our method, **prompt-in-decoder (PiD)**, encodes the input once and decodes output in parallel, boosting both training and inference efficiency by avoiding duplicate input encoding, thereby reducing the decoder's memory footprint. We achieve computation reduction that roughly scales with the number of subtasks, gaining up to 4.6x speed-up over state-of-the-art models for dialogue state tracking, summarization, and question-answering tasks with comparable or better performance.

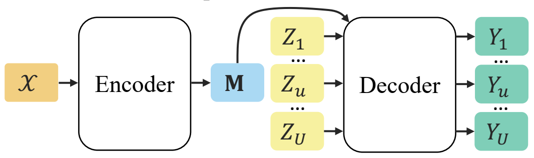
## 1 Introduction

The transformer architecture (Vaswani et al., 2017) is the backbone of many successful NLP models, but they have high costs in computation resources and latency. To reduce costs, researchers have investigated multiple approaches, including model compression (e.g., distillation, Hinton et al., 2015; Gou et al., 2021; Udagawa et al., 2023; quantization, Zadeh et al., 2020; Dettmers et al., 2022; Yao et al., 2022; Dettmers et al., 2023; and mixture of experts Kudugunta et al., 2021), model architecture modifications (e.g., sparse attention, Roy et al., 2021; Liu et al., 2022; multi-query attention, Shazeer, 2019; grouped-query attention, Ainslie et al., 2023), speculative decoding with a smaller model (Leviathan et al., 2023) and GPU kernel optimizations (Dao et al., 2022; Ye et al., 2024a).

Our work focuses on the complementary approach of improving training and inference efficiency in encoder-decoder models on tasks that



a Prompt-in-encoder (PiE).



b (Ours) Prompt-in-decoder (PiD).

Figure 1: Given a task where a single input document $\mathcal{X}$ is used to generate multiple outputs $Y_u$ associated with different prompts $Z_u$, PiE creates unique encodings $\mathbf{M}_u$ for every prompt $Z_u$. In contrast, PiD uses a single shared $\mathbf{M}$ for each prompt. and thus requires less memory access and has higher computational efficiency.

involve multiple queries over the same document (or dialogue). These tasks include scenarios where multiple users are querying the same document with different requests (e.g., question answering, Ye et al., 2024b; Juravsky et al., 2024), as well as scenarios where it is useful to decompose a complex task into simpler subtasks (e.g., abstractive summarization of long/multiple documents, Gidiotis and Tsoumakas, 2020; Meng et al., 2021; Zhang et al., 2022; or information extraction, Lu et al., 2023).

In the multi-user question-answering scenario, decoupling questions and the corresponding document allows reusing the shared document embeddings for questions from different users in a decoder-only model. The shared embeddings can significantly reduce computation during training and inference. The large amount of document embeddings can be can be efficiently compressed and stored (Cao et al., 2023). For summarization and information extraction, decomposing a long target output into multiple shorter sequences mitigates attention degeneration issues (Fu et al., 2023;

Zhou et al., 2023), leading to a boost in accuracy and efficiency. However, existing methods put the prompts in the encoder, resulting in a higher computation cost. In this context, we propose **prompt-in-decoder (PɪD)** an encode-once, decode-in-parallel strategy that avoids duplicate encoding costs by sharing inputs and increases decoding efficiency by reducing memory access.

We demonstrate the effectiveness of our decoding strategy through experiments on a range of tasks with short and long inputs: dialogue state tracking, abstractive medical dialogue summarization, and extractive medical question answering. Our models achieve comparable or higher performance (98-101%) than the current state of the art. At the same time, we observe a 2-10x computation reduction, depending on the number of subtasks, and up to 4.6x speed-up for shorter subtask outputs.

In summary, the main contribution of this work is a new, more efficient decoding strategy for multi-query tasks, validated on several tasks. The associated training/inference code and new models will be released with the published work.

## 2 Encoder-Decoder Framework

The encoder-decoder is a general framework that has been used to address a wide variety of problems in NLP. Given an input word sequence, a desired result is obtained by first encoding the input and then iteratively generating an output word sequence. In general-purpose models, the input $\mathcal{X}$ is optionally combined with a prompt $\mathcal{Z}$ that specifies the task: $\mathcal{Y} = \text{decoder}(\text{encoder}(\mathcal{X}, \mathcal{Z}))$. The input $\mathcal{X}$ is any form of text, e.g., a sentence, article, or transcript of a conversation, and $\mathcal{Z}$ can be an instruction or a question. The output $\mathcal{Y}$ could be information extracted from an article, a summary of a conversation, or a response to a question. State-of-the-art encoder-decoder systems are built on transformers. This section overviews the general framework to introduce notation and set up the multi-subtask inference problem that we address.

### 2.1 Multi-Prompt Decoding

In this paper, we tackle tasks that can be framed in terms of multiple prompts over the same input $\mathcal{X}$. Specifically, the output is a list of subtasks (or answers) $\mathcal{Y}$, where each subtask/answer is $Y_u$, e.g., $\mathcal{Y} = (Y_1, \ldots, Y_u, \ldots, Y_U)$, and $U$ is the total number of subtasks/answers. Each subtask $Y_u$ is associated with a specific prompt $Z_u$, so $\mathcal{Z} = (Z_1, \ldots, Z_u, \ldots, Z_U)$. The scenario involves running inference multiple times to generate $Y_u$, $Y_u = \text{decoder}(\text{encoder}(\mathcal{X}, Z_u))$, then combining all outputs $Y_u$ to form the final $\mathcal{Y}$. We refer to this as the **prompt-in-encoder (PɪE)** approach. Figure 1a illustrates how PɪE tackles a single instance ($\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$). PɪE involves $U$ encodings of $\mathcal{X}$, one for each prompt $Z_u$.

### 2.2 Encode Once and Decode in Parallel

To avoid the redundant encoding of $\mathcal{X}$ in PɪE and improve inference efficiency when decoding $Y_u$, we propose placing prompt $Z_u$ in the decoder, allowing us to encode $\mathcal{X}$ once and decode $Y_u$ in parallel. We refer to this method as **prompt-in-decoder (PɪD)**. By moving $Z_u$ from the encoder (in PɪE) to the decoder, the encoder only encodes $\mathcal{X}$ once, generating a single sequence of embeddings that is reused throughout the decoding process for each prompt $Z_u$. As shown in Figure 1b, $\mathcal{X}$ is only encoded once, and the embeddings $\mathbf{M}$ are reused for $U$ prompts during decoding to generate all subtask outputs $(Y_1, \ldots, Y_u, \ldots, Y_U)$. Formally, the equation can be represented as $Y_u = \text{decoder}(\text{encoder}(\mathcal{X}), Z_u)$.

## 3 Performance Analysis

In this section, we show how the PɪD model improves inference efficiency over the PɪE model by quantifying memory access and the number of arithmetic operations.

### 3.1 Operational Intensity

Memory bandwidth peak performance per second (byte/s) and the number of floating-point operations per second (FLOP/s) are used to compute the operational intensity:

$$\text{Operational Intensity} = \frac{\text{FLOP/s}}{\text{byte/s}} = \frac{\text{\# operations}}{\text{memory access}},$$

which provides a measure for the hardware efficiency of operations (Williams et al., 2009).

To carry out calculations, accelerators must access and move data between global memory and registers, which can be a bottleneck because modern hardware accelerators such as GPUs/TPUs often have significantly greater capacity for computations compared to memory bandwidth. For example, an NVIDIA A100 GPU (Choquette et al., 2021) has an operation capacity of 312 Tera FLOP/s versus a memory bandwidth of 2 Giga byte/s. The

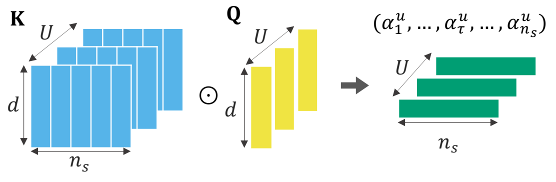
|  | Encoder's Self-attention |  | Decoder's Self-attention |  | Decoder's Cross-attention |  |
|---|---|---|---|---|---|---|
| Model | Memory | Operations | Memory | Operations | Memory | Operations |
| PIE-T5 | $Ubn_sd + d^2$ | $Ubn_sd^2$ | $Ubn_t^2d + n_td^2$ | $Ubn_td^2$ | $Ubn_sn_td + Ubn_td + n_td^2$ | $Ubn_td^2$ |
| PID-T5 | $bn_sd + d^2$ | $bn_sd^2$ | $Ubn_t^2d + n_td^2$ | $Ubn_td^2$ | $bn_sn_td + Ubn_td + n_td^2$ | $Ubn_td^2$ |

Table 1: Inference computation comparison between PIE and PID, where $U$, $b$, $n_s$, $n_t$, $d$ are the number of prompts, batch size, input source length, output target length, and hidden size, respectively.

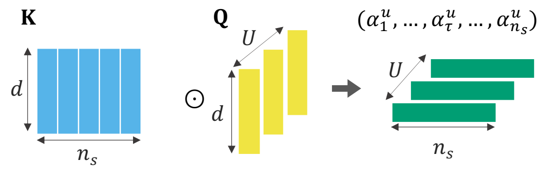
attainable FLOP/s of a device is determined by

$$\text{Attainable FLOP/s} = \min(\text{Peak FLOP/s}, \underbrace{\text{Operational Intensity}}_{\text{\# operations per byte}} \cdot \underbrace{\text{Peak Memory Bandwidth}}_{\text{bytes per second}}),$$

where the peak memory bandwidth is fixed and the peak FLOP/s is the maximum arithmetic operations the accelerator is capable of performing. If the operational intensity is too low, the accelerator idles, waiting for data to move to registers instead of running computations. This often occurs in models where memory access is more intensive than arithmetic operations, i.e., incremental decoding in transformers (Shazeer, 2019). By decreasing memory access, the operational intensity is increased, i.e., efficiency improves.


K
U
d
$n_s$
Q
U
d
$(\alpha_1^u, \ldots, \alpha_\tau^u, \ldots, \alpha_{n_s}^u)$
U
$n_s$

a Cross-attention of PIE.


K
d
$n_s$
Q
U
d
$(\alpha_1^u, \ldots, \alpha_\tau^u, \ldots, \alpha_{n_s}^u)$
U
$n_s$

b Cross-attention of PID.

Figure 2: An illustration of cross-attention dot product operations ($\mathbf{QK}^\top$ in Equation 1) for PIE and PID for a single inference step. $U$, $d$, $n_s$ are the number of prompts, hidden layer dimension, and input length, respectively. $\odot$ is the dot product operation, and the resulting scalars of $\mathbf{QK}^\top$ are $\alpha_\tau^u$, where $\tau = \{1, \ldots, n_s\}$, at the decoding step $\tau$ w.r.t. the prompt $Z_u$.

## 3.2 Multi-head Attention in Transformer

Transformers (Vaswani et al., 2017) have two types of multi-head attention: self-attention and cross-attention. Usually, the attention key and value tensors have the dimension $d/h$, where $d$ is the dimension of input and output vectors and $h$ is the number of attention heads. For simplicity, we consider the operations of all heads together such that the dimension of $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ (query/key/value) in the following section is denoted as $d$.

During attention, the query/key/value vectors can be obtained by projecting the corresponding input vectors $\mathbf{N} \in \mathbb{R}^{n \times d}$ or $\mathbf{M} \in \mathbb{R}^{m \times d}$, where $n$ and $m$ can be either $n_s$ (input source length) or $n_t$ (output target length). More formally, the equations are $\mathbf{Q} = \mathbf{N} \cdot W^Q \in \mathbb{R}^{n \times d}$, $\mathbf{K} = \mathbf{M} \cdot W^K \in \mathbb{R}^{m \times d}$ and $\mathbf{V} = \mathbf{M} \cdot W^V \in \mathbb{R}^{m \times d}$ where the projection matrices are $W^Q, W^K, W^V \in \mathbb{R}^{d \times d}$.

In the self-attention, $\mathbf{N}$ is equivalent to $\mathbf{M}$; hence, $m = n = n_s$ in the encoder or $m = n = n_t$ in the decoder. On the other hand, in the case of the cross-attention, the variable $\mathbf{N} \in \mathbb{R}^{n_t \times d}$ originates from the decoder, while $\mathbf{M} \in \mathbb{R}^{n_s \times d}$ is sourced from the output of the encoder.

The simplified equation of the attention mechanism is represented as follows,

$$\mathbf{O} = \text{softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{d/h}}\right)\mathbf{V} \cdot W^O, \quad (1)$$

where $\mathbf{O} \in \mathbb{R}^{m \times d}$ is the final attention output.

## 3.3 Performance Analysis for PIE and PID

Figure 2 shows the dot product operation in the cross-attention for the two models. In the PIE model, the input is contextualized with each prompt, so the encoder's output tensor differs for each prompt. Thus, at each decoding step $\tau$, $\mathbf{K}$ is read $U$ times to generate $U$ different sets of attention weights $\alpha_\tau^u$ to decode $Y_u$. In contrast, in the PID model, $\mathbf{K}$ is shared across all prompts since the input is encoded independently of the prompts. Thus, at each decoding step $\tau$, the dot product operation in the cross-attention shares and broadcasts $\mathbf{K}$ and computes the dot product of $\mathbf{K}$ and $\mathbf{Q}$, resulting in lower memory access but the same number of arithmetic operations compared to PIE.

We approximate the memory access and operations based on the dominant terms in the self- and cross-attention and ignore the constant terms. For a single input, the memory access of $\mathbf{M}$ and $\mathbf{N}$ is $n_s d$ and $n_t d$. The memory access of the matrices $W^Q$, $W^K$, $W^V$, $W^O$ is $d^2$. The number of operations in both attention mechanisms is dominated by the matrix projections which are used to obtain $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$, and $\mathbf{O}$; thus, the number of operations is approximated as $n_s d^2$ or $n_t d^2$.

The comparisons of memory access and operation counts on different inference components for our PIE and PID implementations are presented in Table 1, explained in further detail below. In the analysis, we assume that a batch of $b$ inputs with the same set of $U$ subtasks are processed together. The encoder input length and the decoder output length differ depending on whether the prompt is in the decoder. For simplicity, the analysis also assumes that the prompt terms are negligible; Appendix A provides the detailed justification.

**Encoder's self-attention.** In PIE, to run inference on a single instance, the model encodes $U$ prompts ($Z_u$) with input ($\mathcal{X}$). Considering a batch with $b$ instances, PIE takes $Ubn_s d + d^2$ for memory access and $Ubn_s d^2$ for the number of operations. In contrast, PID only encodes the input once, so the memory access and the number of operations remain $bn_s d + d^2$ and $bn_s d^2$. Thus, the encoders of both models have similar operational intensity, but PIE requires more memory access and more arithmetic computations.

**Decoder's self-attention.** In both PIE and PID, the decoder computes the self-attention for $U$ prompts. For each decoding step, the memory access and the number of operations are $Ubn_t d$ and $Ubd^2$. Thus, for $n_t$ steps, the resulting memory access and the number of operations are $Ubn_t^2 d$ and $Ubn_t d^2$. In this case, PIE and PID have roughly the same operational intensity.

**Decoder's cross-attention.** Cross-attention dominates the inference cost. For each decoding step, we load the encoded input embeddings $\mathbf{M} \in \mathbb{R}^{m \times d}$, where $m = n_s$, from the encoder for each $(\mathcal{X}, Z_u)$ input for PIE and $\mathcal{X}$ for PID. For PIE, the resulting $b$ $(\mathbf{M}_1, \ldots, \mathbf{M}_U)$ is fed into the decoder's cross-attention for $b$ instances with $U$ prompts. On the other hand, PID shares all $\mathbf{M}$ for all $U$ prompts of each instance, resulting in feeding $b$ $\mathbf{M}$ to the decoder. Therefore, for each step, the memory access for loading encoded $\mathbf{M}$ is $Ubn_s d$ and $bn_s d$ for PIE and PID, respectively. The memory access of loading $\mathbf{Q}$ and $\mathbf{O}$ is $Ubd$. Loading projection matrices takes $d^2$. We multiply all memory access cost by a factor of $n_t$ steps.

## 4 Datasets & Metrics

### 4.1 Datasets & Task Performance Metrics

**Dialogue State Tracking (DST).** Multi-domain Wizard-of-Oz dataset (MultiWoZ; Budzianowski et al., 2018) is a task-oriented dialogue dataset. We selected the most recent version of MultiWoZ 2.4 (Ye et al., 2022b) due to its refined validation and test set annotations. For comparison to other work, joint goal accuracy (JGA) is adopted as the evaluation metric. The input $\mathcal{X}$ is the dialogue history, $\mathcal{Y}$ is the dialogue state, the subtask prompts $Z_u$ are the domain-slot name, and the associated outputs $Y_u$ are slot values.

**Summarization.** We use ACI-Bench (Yim et al., 2023), a dataset containing clinical notes associated with conversations between doctors and patients. The clinical notes have structured output with distinct sections. We use ROUGE-L score (Lin, 2004), denoted as R-L, to evaluate models. The input $\mathcal{X}$ is the doctor-patient dialogue, $\mathcal{Y}$ is the full clinical note, the subtask prompts $Z_u$ are section indicators, and the associated outputs $Y_u$ are section notes.

**Question Answering.** RadQA (Soni et al., 2022) is an extractive question-answering dataset on radiology reports with 3k questions posed by experts. A single report can have multiple questions. We use exact match (EM) as our evaluation metric. The input $\mathcal{X}$ is the radiology report, the subtask prompts $Z_u$ are specific questions, and the associated outputs $Y_u$ are extracted responses.

### 4.2 Efficiency Metrics

**Floating point operations (FLOPs)** refer to the number of arithmetic operations required for model inference, i.e., the computational complexity.[1] Note that FLOP reduction may not correlate with wall-clock speed, as it tends to ignore overheads from memory access (IO) (Dao et al., 2022).

**Latency.** To account for extra IO costs, e.g., GPU memory bandwidth, we also report latency, which measures the wall clock time for a single instance inference. Specifically, we report the average time for a single instance inference, where instances are processed sequentially.

[1]We use `calflops` (Ye, 2023) to compute FLOPs.

**Latency w/ Batching.** For real applications, multiple instances are computed in a batch fashion to fully utilize the computing device, especially for cloud serving. We thus report the average time for a single instance, where a batch of instances is computed simultaneously.

To assess the FLOPs and latency, we randomly chose 512 samples from MultiWoZ 2.4 test set (due to the large test set) and used the full test sets for ACI-Bench and RadQA. We report the average latency and the average latency w/ batching at the optimal batch size.

## 5 Experiments & Results

### 5.1 Compared Systems

**T5** Raffel et al. (2020) is adopted as the encoder-decoder model in all experiments. We use T5-base and T5-large models from HuggingFace for MultiWoZ 2.4 and ACI-Bench. For RadQA, we follow the previous work (Lehman et al., 2023) and use a pretrained clinical T5. We denote base and large models in following tables as T5$_{\text{base}}$ and T5$_{\text{large}}$. We use T5 as the backbone to compare two different subtasking strategies: prompt-in-encoder (PIE) and our proposed prompt-in-decoder (PID); thus the model size keeps the same as the standard T5.

**LLaMA2** Touvron et al. (2023) is a popular open decoder-only language model. Due to computation limitations, we adopt low-rank adaptation (LoRA; Hu et al., 2022a) to efficiently finetune LLaMA2 7B, resulting in approximately 70M trainable parameters with a rank of 16.

**Current state-of-the-art models.** We report the current best published results for in-context learning and full finetuning for every dataset. The source papers for the results of each dataset are documented in the caption of Table 4.

### 5.2 Training Procedure

The standard fine-tuned T5 and LLaMA2 data is represented as $(\mathcal{X}, \mathcal{Y})$, i.e. no prompts are used. For PIE-T5, the data is $(\mathcal{X}, Y_u, Z_u)$, since $\mathcal{X}$ is separately contextualized with each subtask prompt $Z_u$ and a subtask output $Y_u$, effectively increasing the dataset size by a factor of $U$ and substantially extending the time required for training. In contrast, PID-T5 has the flexibility to be trained using either data representation, as it uses shared inputs. We choose $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ for PID-T5 because it is more efficient to put all output $\mathcal{Y}$ from the same shared input $\mathcal{X}$ in the same batch to save encoding processing time during gradient update. The model selection criterion is the highest score on the validation set. Hyperparameters can be found in Appendix B.

| Model | MultiWoZ 2.4 JGA ↑ | ACI-Bench R-L ↑ | RadQA EM ↑ |
|---|---|---|---|
| T5$_{\text{base}}$ | 71.5 | 47.9 | – |
| PIE-T5$_{\text{base}}$ | 76.3 | 53.8 | 53.7 |
| PID-T5$_{\text{base}}$ | 75.5 | 54.0 | 52.4 |
| T5$_{\text{large}}$ | 72.5 | 52.5 | – |
| PIE-T5$_{\text{large}}$ | 77.5 | 54.7 | 55.4 |
| PID-T5$_{\text{large}}$ | 76.5 | 55.2 | 54.6 |
| LLaMA2 $_{\text{LoRA}}$ | 66.1 | 53.8 | 54.4 |

Table 2: Task performance comparison between the baseline models and PID-T5 over three public evaluation tasks. The sizes of T5$_{\text{base}}$ and T5$_{\text{large}}$ are 220M and 770M, respectively. We use LLaMA2 7B and the number of trainable parameters is about 70M.

### 5.3 Results

**Task performance.** Table 2 presents the task performance results for three different full finetuning scenarios (T5, PIE-T5, and PID-T5), as well as a LoRA finetuned LLaMA2 language model. The results of MultiWoZ 2.4 and ACI-Bench indicate that subtasking and multi-prompt decoding help, since both PIE-T5 and PID-T5 outperform the standard T5 and LLaMA2 models. Although larger model size often comes with better performance in moving from T5$_{\text{base}}$ to T5$_{\text{large}}$, there are mixed results for LLaMA2 on these domains. One possible reason could be that the number of trainable parameters is fewer than that of full finetuning. More importantly, similar or greater gains in performance are obtained with smaller, more efficient models that leverage task structure.

**Computation efficiency.** Since PIE-T5 and PID-T5 achieve better results than standard T5, we compare PIE-T5 and PID-T5 in Table 3. Regarding computational efficiency, measured in FLOPs, PID-T5 significantly outperforms PIE-T5 in reducing the number of arithmetic operations because it processes each input only once. PID-T5 achieves superior speed-up in both single-instance and batching scenarios across three datasets and two model sizes, while obtaining similar performance (98-101%) to PIE-T5. Additionally, PID-T5 offers greater reductions in computational costs (2-10x) and further accelerates efficiency when dealing with a larger number of subtasks—for example, managing 30

| | MultiWoZ 2.4 | | | | ACI-Bench | | | | RadQA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | JGA ↑ | FLOPs ↓ | Sp↑ | $Sp^{batch}$ ↑ | R-L ↑ | FLOPs ↓ | Sp↑ | $Sp^{batch}$ ↑ | EM ↑ | FLOPs ↓ | Sp↑ | $Sp^{batch}$ ↑ |
| PIE-$T5_{base}$ | 76.3 | 1.0x | 1.0x | 1.0x | 53.8 | 1.0x | 1.0x | 1.0x | 53.7 | 1.0x | 1.0x | 1.0x |
| PID-$T5_{base}$ | 75.5 | **0.1x** | **1.9x** | **4.6x** | 54.0 | **0.4x** | **1.1x** | **1.1x** | 52.4 | **0.6x** | **1.3x** | **1.3x** |
| PIE-$T5_{large}$ | 77.5 | 1.0x | 1.0x | 1.0x | 54.7 | 1.0x | 1.0x | 1.0x | 55.4 | 1.0x | 1.0x | 1.0x |
| PID-$T5_{large}$ | 76.5 | **0.1x** | **2.8x** | **4.2x** | 55.2 | **0.4x** | 1.0x | **1.5x** | 54.6 | **0.5x** | **2.8x** | **1.3x** |
| LLaMA2 $_{LoRA}$ | 66.1 | 0.7x | 0.2x | 0.5x | 53.8 | 4.8x | 0.3x | 0.3x | 54.4 | 26.8x | 0.2x | 0.1x |

Table 3: Task performance and inference efficiency comparison between PIE-T5, PID-T5 and LLaMA2 over three public evaluation tasks. Sp and $Sp^{batch}$ represent the relative speed-up in single-instance and batching scenarios computed on NVIDIA A100. We present the relative ratios of FLOPs, Sp and $Sp^{batch}$ compared to PIE-$T5_{base}$ or PIE-$T5_{large}$, across other models. Overall, PID-T5 achieves best computation efficiency across all tasks and achieves comparable task performance on MultiWoZ 2.4 and RadQA and better task performance on ACI-Bench.

slots in MultiWoZ 2.4 versus 4 sections in ACI-Bench and 2-6 questions in RadQA.

**Comparison between PID-$T5_{large}$ and state-of-the-art models.** Table 4 illustrates the comparison between our method, PID-$T5_{large}$, and existing state-of-the-art approaches. Our PID-$T5_{large}$ outperforms in-context learning methods on all three datasets with much smaller models. Compared to full fine-tuning, our model outperforms on MultiWoZ 2.4 and ACI-Bench, but underperforms on RadQA. This discrepancy could be attributed to the fact that the prompts in RadQA are less structured compared to the fixed types of prompts in tasks with structured outputs.

We omit the inference cost of state-of-the-art models since access to the language models for in-context learning is restricted to API calls, and some fine-tuned models' checkpoints are unavailable. Nevertheless, we can assume that the inference cost exceeds that of the proposed PID-T5 due to the larger model sizes or inference techniques employed. The $T5_{XXL}$ inference in MultiWoZ 2.4 follows the standard T5, while the inference approaches for $Bart_{large}$ and $T5_{large}$ in ACI-Bench and RadQA are consistent with those in PIE-T5.

**Comparison between in-context learning and finetuned models in the low-resource setting.** Figure 3 shows the comparison between in-context learning and full finetuned models. We use the same 1%, 5% and 10% training set provided in Hu et al. (2022b). PID-$T5_{base}$ surpasses $Codex_{davinci}$ when the 5% training ($\approx$ 374 examples) is available with 0.1% model size.[2] The result suggests that the small, finetuned model is still useful when a reasonable amount of training data is available.

[2]Some papers mention that $Codex_{davinci}$ is 175B, but OpenAI does not officially confirm that.

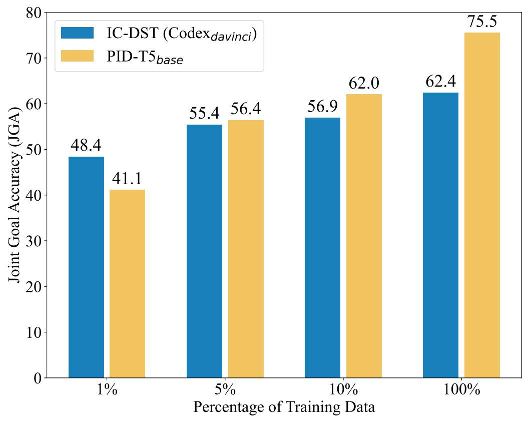


Figure 3: Comparison between in-context learning and full finetuning on MultiWoZ 2.4 test set. The JGA scores are reported at 1%, 5%, 10% and 100% of training data. We use the 3 random splits of the training data provided by IC-DST(Hu et al., 2022b).

**Efficiency under the batching scenario.** The efficiency of batching is reflected in latency w/ batching. Batching instances with mixed output lengths requires padding all outputs to the same length. The additional padding tokens during the decoding phase leads to increased computational overhead for PIE-T5 and PID-T5 compared to T5. The padding issue will be amplified in PIE-T5 and PID-T5 when the number of prompts $U$ is relatively large, e.g., 30 slots in MultiWoZ 2.4. Figure 4 shows the computational efficiency and the task performance. PID-T5 outperforms the standard T5 model and achieves similar task performance to PIE-T5, while maintaining the same scale of FLOPs as the standard T5. Furthermore, the latency w/ batching of PIE-T5 is more sensitive to model size than either T5 or PID-T5.

**Training efficiency.** Table 5 shows the training costs associated with models and training strategies described in subsection 5.2. PIE-$T5_{base}$ incorpo-

| | MultiWoZ 2.4 | | | ACI-Bench | | | RadQA | | |
|---|---|---|---|---|---|---|---|---|---|
| Learning Method | Model | Size↓ | JGA ↑ | Model | Size↓ | R-L ↑ | Model | Size↓ | EM ↑ |
| In-context learning sota | Codex$_{\text{davinci}}$ | 175B | 62.4 | GPT4-32k | 1760B | 54.3 | GPT3 | 175B | 36.2 |
| Full finetuning sota | T5$_{\text{XXL}}$ | 11B | 75.9 | Bart$_{\text{large}}$ | 4×406M | 48.6 | T5$_{\text{large}}$ | **770M** | **55.0** |
| Our PID | T5$_{\text{large}}$ | **770M** | **76.5** | T5$_{\text{large}}$ | **770M** | **55.2** | T5$_{\text{large}}$ | **770M** | 54.6 |

Table 4: We choose previous state-of-the-art (sota) generative models from the literature with the most comparable model sizes. In-context learning sota results for MultiWoZ, ACI-Bench, and RadQA are reported in the studies (Hu et al., 2022b; Yim et al., 2023; Lehman et al., 2023), respectively. For full finetuning sota, the results are reported in the studies (Zhao et al., 2022; Yim et al., 2023; Lehman et al., 2023).Yim et al. (2023) use four Bart$_{\text{large}}$ models (one for each section), resulting in quadruple the size of a single Bart$_{\text{large}}$ (406M).

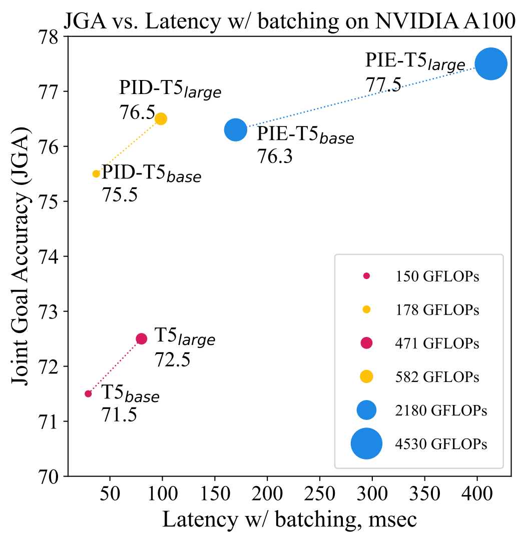


Figure 4: Comparison of joint goal accuracy (JGA) and latency w/ batching on the MultiWoZ 2.4 test set. Models positioned in the upper left corner indicate superior task performance coupled with faster decoding. A larger bubble indicates the model requires more FLOPs to complete a instance.

| | Data | Training FLOPs ↓ | JGA ↑ |
|---|---|---|---|
| T5$_{\text{base}}$ | $(\mathcal{X}, \mathcal{Y})$ | $1.5 \times 10^{17}$ | 71.5 |
| PIE-T5$_{\text{base}}$ | $(\mathcal{X}, Y_u, Z_u)$ | $5.0 \times 10^{18}$ | 76.3 |
| PID-T5$_{\text{base}}$ | $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ | $1.2 \times 10^{17}$ | 75.5 |

Table 5: Comparison of training cost across models and different training procedure on MultiWoZ 2.4. The training computational costs are reported in FLOPs, and the JGA scores are reported on the test set.

rates prompts within its encoder, necessitating the enumeration of all prompts $\mathcal{Z} = (Z_1, \ldots, Z_U)$ for every input $\mathcal{X}$ during both training and testing phases. Consequently, PIE-T5$_{\text{base}}$ requires more FLOPs, i.e., longer training duration, as the total number of training samples is increased by a factor of $U$. Conversely, PID-T5$_{\text{base}}$ allows the reuse of the same input across all prompts, keeping the total number of training examples the same as T5. PID-T5$_{\text{base}}$ not only maintains a comparable JGA score and has efficient inference but also reduces training costs.

| | JGA↑ | L$_{\text{A100}}$ ↓ | Sp ↑ | L$_{\text{2080Ti}}$ ↓ | Sp ↑ |
|---|---|---|---|---|---|
| PIE-T5$_{\text{base}}$ | 76.3 | 146 | 1.0x | 209 | 1.0x |
| PID-T5$_{\text{base}}$ | 75.5 | 78 | **1.9x** | 91 | **2.3x** |
| PIE-T5$_{\text{large}}$ | 77.5 | 413 | 1.0x | 625 | 1.0x |
| PID-T5$_{\text{large}}$ | 76.5 | 147 | **2.8x** | 163 | **3.8x** |

Table 6: Comparison of latency (measured in msec) between different levels of GPUs on MultiWoZ 2.4. L$_{\text{A100}}$ and L$_{\text{2080Ti}}$ stand for latency on NVIDIA A100 and RTX 2080Ti, respectively. Sp represent the relative speed-up relative to PIE-T5$_{\text{base}}$ or PID-T5$_{\text{large}}$.

**Latency on different levels of GPUs.** Table 6 illustrates that our PID-T5$_{\text{base}}$ surpasses PIE-T5$_{\text{base}}$ in efficiency under a single-instance scenario. This difference becomes more pronounced when using a consumer-grade GPU, e.g., NVIDIA RTX 2080Ti, and inferencing on the larger model.

| Model | Subtask | JGA ↑ | FLOPs ↓ | Sp ↑ |
|---|---|---|---|---|
| T5$_{\text{base}}$ | All | 71.5 | 1.0x | 1.0x |
| PID-T5$_{\text{base}}$ | Domain | 72.5 | 2.3x | 12.3x |
| PID-T5$_{\text{base}}$ | Domain-Slot | 75.5 | 2.5x | 5.9x |

Table 7: Comparison between different subtask scales, i.e., 30 domain slots or 5 domains, on MultiWoZ 2.4. Sp represents the relative speed-up in latency computed on NVIDIA A100. The model with the domain subtask predicts all active slot values in that domain.

**Effect of different subtask granularity.** In MultiWoZ 2.4, T5's output can be broken down into subtasks according to either domains (where the output is a sequence of observed slots and their values) or domain-slot pairs (where the output is the slot value). Each domain or domain-slot pair

| Model | All | Section (ROUGE-L) 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $T5_{base}$ | 47.9 | 34.3 | 28.8 | 28.4 | 17.9 |
| PIE-$T5_{base}$ | 53.8 | **36.9** | 57.2 | 50.9 | 35.4 |
| PID-$T5_{base}$ | **54.0** | 36.6 | **57.7** | **58.9** | **35.9** |

Table 8: Comparison between T5, PIE-T5 and PID-T5 on ACI-Bench test 1 set.

is associated with a individual prompt. As demonstrated in Table 7, the system PID-$T5_{base}$ reveals that employing multi-prompt decoding can enhance both the inference speed and the joint goal accuracy (JGA) score, regardless of the granularity of the subtask units. While utilizing domains as subtask units leads to more speed-up, the use of slots as subtask units yields the best JGA.

**Effect of subtasking for long output.** In ACI-Bench, We adopt the structure proposed by (Yim et al., 2023) for organizing the summary output into four distinct parts: subjective, objective examination, objective findings, and assessment with planning. The mean section lengths are specified as 285, 98, 35, 254 tokens, respectively. The first section details the patient's medical history, while the fourth section focuses on assessment and planning; these sections surpass the second and third sections in terms of length.

Table 8 reveals that the standard $T5_{base}$ model underperforms with longer outputs, especially when the generation reaches the end of the sequence, i.e., performance of later sections are much worse than for the other models. Both PIE-$T5_{base}$ and PID-$T5_{base}$ models demonstrate improved performance with subtasking, allowing the model to "focus" on one subtask at a time.

## 6 Related Work

**Reduce model size.** Reducing memory bandwidth bottleneck can be accomplished by quantizing model parameters and, in some cases, activations (Dettmers et al., 2022; Zeng et al., 2023; Zhao et al., 2023). This compression of models decreases the amount of data transfer and also lowers the total memory consumption, enabling the use of larger and more effective batch sizes. Moreover, it is possible to distill models into a smaller student model, which is cheaper for inference (Hinton et al., 2015; Gou et al., 2021; Hsieh et al., 2023).

**Reduce cross-attention overhead.** Previous work has found that data movement is often a constraining factor for computations on modern devices. Shazeer (2019) shows that autoregressive transformers are particularly bandwidth bound during inference, and proposes multi-query attention (MQA) as a partial solution. Ainslie et al. (2023) introduce grouped-query attention (GQA), a generalization of MQA which uses an intermediate number (more than one, less than the number of query heads) of key-value heads. GQA achieves quality close to original multi-head attention (MHA) with comparable speed to MQA.

**Parallel decoding.** One method to address the sequential decoding performance bottleneck is to decode in parallel. Ning et al. (2023) use off-the-shelf models and enable parallel decoding by using a two-step strategy: use the LM to generate a skeleton outline, then fill in the outline with parallel decoding requests. Different from ours, their strategy does not involve finetuning and focuses on decoder-only models. Another approach is to decouple the attention over a shared prefix and unique suffix and store the shared key-value cache, reducing redundant memory access and enabling multi-user parallel decoding (Ye et al., 2024b; Juravsky et al., 2024). Their method focuses on kernel-level speedups and decoder-only models.

Our method is compatible with the aforementioned efficiency techniques, theoretically leading to further efficiency gains when used in concert.

## 7 Conclusion

We study settings arising in NLP where multiple text queries are applied to the same document or dialogue. The subtasking approach allows a model to individually address components of the main task, leading to improved task performance and accelerating decoding by enabling tasks to be completed in parallel. The strategy of moving the prompt from the encoder to the decoder allows the PID method to reduce computational costs by encoding the input just once and then reusing it for multiple prompts, which further speeds up inference time while either maintaining or improving task performance. Our approach achieves higher efficiency and comparable accuracy to existing approaches, which is particularly valuable in scenarios where computational resources are scarce.

## 8 Limitations

The types of tasks where our method is applicable are currently limited because we require tasks with a shared input document. The subtasking strategies in our datasets were designed by humans, e.g., according to structured output sections. Instead of using human-designed subtasking rules, a potential avenue for exploration is to allow a model to learn how to subtask, which can additionally make more tasks possible. While our subtasking experiments use only encoder-decoder models, our strategy of sharing an embedding and decomposing a task should work with decoder-only models, as in Khot et al. (2023), but experimental analysis is left to future work.

## 9 Ethical Considerations

Our research contributes to the extensive field of studies focused on transformer-based generative language models. This study targets faster and more efficient inference capabilities. This advance enhances practical applications such as dialogue state tracking, summarization, and question answering for serving users. However, it also brings with it the inherent risks associated with all generative models.

## References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, Singapore. Association for Computational Linguistics.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026, Brussels, Belgium. Association for Computational Linguistics.

Qingqing Cao, Sewon Min, Yizhong Wang, and Hannaneh Hajishirzi. 2023. Btr: Binary token representations for efficient retrieval augmented language models. *arXiv preprint arXiv:2310.01329*.

Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*.

Michiel de Jong, Yury Zemlyanskiy, Joshua Ainslie, Nicholas FitzGerald, Sumit Sanghai, Fei Sha, and William Cohen. 2023. FiDO: Fusion-in-decoder optimized for stronger performance and faster inference. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 11534–11547, Toronto, Canada. Association for Computational Linguistics.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zihao Fu, Wai Lam, Qian Yu, Anthony Man-Cho So, Shengding Hu, Zhiyuan Liu, and Nigel Collier. 2023. Decoder-only or encoder-decoder? interpreting language model as a regularized encoder-decoder. *arXiv preprint arXiv:2304.04052*.

Alexios Gidiotis and Grigorios Tsoumakas. 2020. A divide-and-conquer approach to the summarization of long documents. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:3029–3040.

Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017, Toronto, Canada. Association for Computational Linguistics.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022a. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Yushi Hu, Chia-Hsuan Lee, Tianbao Xie, Tao Yu, Noah A. Smith, and Mari Ostendorf. 2022b. In-context learning for few-shot dialogue state tracking. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2627–2643, Abu

Dhabi, United Arab Emirates. Association for Computational Linguistics.

Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y Fu, Christopher Ré, and Azalia Mirhoseini. 2024. Hydragen: High-throughput llm inference with shared prefixes. *arXiv preprint arXiv:2402.05099*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.

Sneha Kudugunta, Yanping Huang, Ankur Bapna, Maxim Krikun, Dmitry Lepikhin, Minh-Thang Luong, and Orhan Firat. 2021. Beyond distillation: Task-level mixture-of-experts for efficient inference. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3577–3599, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Eric Lehman, Evan Hernandez, Diwakar Mahajan, Jonas Wulff, Micah J Smith, Zachary Ziegler, Daniel Nadler, Peter Szolovits, Alistair Johnson, and Emily Alsentzer. 2023. Do we still need clinical language models? In *Proceedings of the Conference on Health, Inference, and Learning*, volume 209 of *Proceedings of Machine Learning Research*, pages 578–597. PMLR.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Liu Liu, Zheng Qu, Zhaodong Chen, Fengbin Tu, Yufei Ding, and Yuan Xie. 2022. Dynamic sparse attention for scalable transformer acceleration. *IEEE Transactions on Computers*, 71:3165–3178.

Bo-Ru Lu, Nikita Haduong, Chia-Hsuan Lee, Zeqiu Wu, Hao Cheng, Paul Koester, Jean Utke, Tao Yu, Noah A. Smith, and Mari Ostendorf. 2023. Dialgen: Collaborative human-lm generated dialogues for improved understanding of human-human conversations.

Rui Meng, Khushboo Thaker, Lei Zhang, Yue Dong, Xingdi Yuan, Tong Wang, and Daqing He. 2021. Bringing structure into summaries: a faceted summarization dataset for long scientific documents. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 1080–1089, Online. Association for Computational Linguistics.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Huazhong Yang, and Yu Wang. 2023. Skeleton-of-thought: Large language models can do parallel decoding. *arXiv preprint arXiv:2307.15337*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient Content-Based Sparse Attention with Routing Transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68.

Noam M. Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *ArXiv*, abs/1911.02150.

Sarvesh Soni, Meghana Gudala, Atieh Pajouhi, and Kirk Roberts. 2022. RadQA: A question answering dataset to improve comprehension of radiology reports. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 6250–6259, Marseille, France. European Language Resources Association.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models, 2023. *URL https://arxiv.org/abs/2307.09288*.

Takuma Udagawa, Aashka Trivedi, Michele Merler, and Bishwaranjan Bhattacharjee. 2023. A comparative analysis of task-agnostic distillation methods for compressing transformer language models. In *Conference on Empirical Methods in Natural Language Processing*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183.

Fanghua Ye, Yue Feng, and Emine Yilmaz. 2022a. ASSIST: Towards label noise-robust dialogue state tracking. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2719–2731, Dublin, Ireland. Association for Computational Linguistics.

Fanghua Ye, Jarana Manotumruksa, and Emine Yilmaz. 2022b. MultiWOZ 2.4: A multi-domain task-oriented dialogue dataset with essential annotation corrections to improve state tracking evaluation. In *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 351–360, Edinburgh, UK. Association for Computational Linguistics.

Xiaoju Ye. 2023. calflops: a flops and params calculate tool for neural networks in pytorch framework.

Zihao Ye, Lequn Chen, Ruihang Lai, Yilong Zhao, Size Zheng, Junru Shao, Bohan Hou, Hongyi Jin, Yifei Zuo, Liangsheng Yin, and Tianqi Chen Luis Ceze. 2024a. Flashinfer: Kernel library for llm serving. `https://flashinfer.ai/2024/02/02/introduce-flashinfer.html`. Accessed: January 31, 2024.

Zihao Ye, Ruihang Lai, Bo-Ru Lu, Chien-Yu Lin, Size Zheng, Lequn Chen, Tianqi Chen, and Luis Ceze. 2024b. Cascade inference: Memory bandwidth efficient shared prefix batch decoding. `https://flashinfer.ai/2024/01/08/cascade-inference.html`. Accessed: January 31, 2024.

Wen-wai Yim, Yujuan Fu, Asma Ben Abacha, Neal Snider, Thomas Lin, and Meliha Yetisgen. 2023. Aci-bench: a novel ambient clinical intelligence dataset for benchmarking automatic visit note generation. *Scientific Data*, 10(1):586.

Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. 2020. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 811–824. IEEE.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. GLM-130b: An open bilingual pre-trained model. In *The Eleventh International Conference on Learning Representations*.

Yusen Zhang, Ansong Ni, Ziming Mao, Chen Henry Wu, Chenguang Zhu, Budhaditya Deb, Ahmed Awadallah, Dragomir Radev, and Rui Zhang. 2022. Summ$^n$: A multi-stage summarization framework for long input dialogues and documents. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1592–1604, Dublin, Ireland. Association for Computational Linguistics.

Jeffrey Zhao, Raghav Gupta, Yuan Cao, Dian Yu, Mingqiu Wang, Harrison Lee, Abhinav Rastogi, Izhak Shafran, and Yonghui Wu. 2022. Description-driven task-oriented dialog modeling. *arXiv preprint arXiv:2201.08904*.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2023. Atom: Low-bit quantization for efficient and accurate llm serving. *arXiv preprint arXiv:2310.19102*.

Sitong Zhou, Meliha Yetisgen, and Mari Ostendorf. 2023. Building blocks for complex tasks: Robust generative event extraction for radiology reports under domain shifts. In *Proceedings of the 5th Clinical Natural Language Processing Workshop*, pages 344–357, Toronto, Canada. Association for Computational Linguistics.

## A Detailed Performance Analysis

Higher operational intensity bring more efficient matrix computation in modern accelerators such GPUs/TPUs. In this section, we detail the performance analysis and compare the operational intensity ratios of PIE and PID. To simplify equations, we follow the previous work (Shazeer, 2019; de Jong et al., 2023; Ainslie et al., 2023), using the inverse operational intensity ($\mathcal{R}$) to compare all modules. The lower inverse ratio indicates higher operational intensity, hence better performance.

We discuss memory access and number of floating-point operations of encoder's self-attention, decoder's self-attention and decoder's cross-attention for PIE and PID, respectively. We denote $n_s$, $n_t$ and $n_p$ as input length, output length and prompt length. $U$ is number of prompts/subtasks. $d$ is the joint dimension of all heads of key/query/value vectors.

We approximate the memory access and the number of operations based on the dominant terms in the self- and cross-attention and ignore the constant terms. For a single input, the memory access of the key or value tensors $\mathbf{M}$ or $\mathbf{N}$ are $n_s d$ and $n_t d$. The memory access of the projection matrices of key/query/value/output tensors $W^K$, $W^Q$, $W^V$, $W^O$ is $d^2$. As described in Shazeer (2019), the number of operations in both attention mechanisms is dominated by the matrix projections which are used to obtain projected query/key/value/output tensors (**Q/K/V/O**); thus, the number of operations is approximated as $n_s d^2$ or $n_t d^2$.

### A.1 Encoder's self-attention

The inverse operational intensity of PIE's encoder can be written as follows,

$$\mathcal{R}_{\text{PIE}}^{\text{Enc-self}} = \underbrace{Ub(n_s+n_p)d+d^2}_{\text{memory access}} \Big/ \underbrace{Ub(n_s+n_p)d^2}_{\text{\# operations}}$$
$$= \frac{1}{d} + \frac{1}{Ub(n_s+n_p)},$$

where PIE's encoder individually encodes $U$ prompts $(Z_1, \ldots, Z_U)$ with input $\mathcal{X}$. $\mathcal{R}_{\text{PIE}}^{\text{Enc-self}}$ is a low ratio given the fact that $n_s$ is usually an hundred or a thousand tokens and $d$ is usually near a thousand.

Different from PIE's encoder, PID's encoder only encodes input $\mathcal{X}$ and leaves prompts in the decoder. Thus, the memory access is lower than PIE by a factor of $U$ and only the input length $n_s$ is considered. More formally the inverse operational intensity of PID's encoder is denoted as

$$\mathcal{R}_{\text{PID}}^{\text{Enc-self}} = \underbrace{bn_s d+d^2}_{\text{memory access}} \Big/ \underbrace{bn_s d^2}_{\text{\# operations}}$$
$$= \frac{1}{d} + \frac{1}{bn_s}.$$

Again, $n_s$ and $d$ is around a thousand, resulting in $\mathcal{R}_{\text{PID}}^{\text{Enc-self}}$ is also a low ratio. Compared to PIE, PID requires fewer number of operations, saving more memory usage and enabling faster computation.

### A.2 Decoder's self-attention

In PIE, prompts are encoded in the encoder, the decoder only accounts for generating target tokens. The inverse operational intensity of PIE encoder's self-attention is denoted as

$$\mathcal{R}_{\text{PIE}}^{\text{Dec-self}} = \underbrace{Ubn_t^2 d+n_t d^2}_{\text{memory access}} \Big/ \underbrace{Ubn_t d^2}_{\text{\# operations}}$$
$$= \underbrace{\frac{n_t}{d}}_{\text{dominant term}} + \frac{1}{Ub},$$

where $\frac{n_t}{d}$ is the dominant term that causes the issue of slower incremental decoding.

PID's decoder encodes prompts and incrementally generates output tokens. We decouple the analysis of encoding prompts and generating output tokens since the prompts are all given whereas the output tokens are incrementally decoded. The decoder just need to encode the prompts once; as a result, the ratio can be written as

$$\mathcal{R}_{\text{PID}}^{\text{Dec-self,prompt}} = \underbrace{Ubn_p d+d^2}_{\text{memory access}} \Big/ \underbrace{Ubn_p d^2}_{\text{\# operations}}$$
$$= \frac{1}{d} + \frac{1}{Ubn_p}$$

Obviously, the $\mathcal{R}_{\text{PID}}^{\text{Dec-self,prompt}}$ is a low ratio; thus the encoding prompts in the decoder part is efficient. In addition to encoding prompts, PID's decoder needs to generate output tokens. The ratio $\mathcal{R}_{\text{PID}}^{\text{Dec-self,output}}$ is the same as $\mathcal{R}_{\text{PIE}}^{\text{Dec-self}}$.

### A.3 Decoder's cross-attention

In transformer architecture, the decoder's cross-attention is the key issue that cause the computation inefficiency since the inference is incremental and cross-attention needs to read the huge chunk of

| | MultiWoZ 2.4 | ACI-Bench | RadQA |
|---|---|---|---|
| Data | Task oriented | Medical | Medical |
| Input type | Dialogue | Dialogue | Document |
| Task | Dialog state tracking | Summarization | Question Answering |
| # examples | 9887 | 207 | 6148 |
| Input length | $289_{\pm 108}$ | $1725_{\pm 511}$ | $137_{\pm 157}$ |
| Output length | $56_{\pm 26}$ | $693_{\pm 200}$ | $28_{\pm 49}$ |
| Prompt type | Fixed | Fixed | Free-form |
| # prompts | 30 slots or 5 domains | 4 sections | – |

Table 9: Statistics are calculated on the each full data set. Input and output lengths are calculated based on Huggingface T5 tokenizer.

| | MultiWoZ 2.4 | | ACI-Bench | | RadQA | |
|---|---|---|---|---|---|---|
| | PID-T5$_{\text{base}}$ | PID-T5$_{\text{large}}$ | PID-T5$_{\text{base}}$ | PID-T5$_{\text{large}}$ | PID-T5$_{\text{base}}$ | PID-T5$_{\text{large}}$ |
| Batch size | 4 | 1 | 1 | 2 | 4 | 2 |
| Grdient accumulation | 64 | 32 | 16 | 16 | 16 | 32 |
| Effective batch size | 64 | 64 | 32 | 32 | 64 | 64 |
| # epochs | 6 | 4 | 100 | 100 | 50 | 15 |
| Max input length | 1024 | 1024 | 3072 | 3072 | 1024 | 1024 |
| Max output length | 24 | 24 | 1024 | 1024 | 92 | 92 |
| Max prompt length | 8 | 8 | 6 | 6 | 36 | 36 |
| # outputs | 30 | 30 | 4 | 4 | 2-6 | 2-6 |
| # beams | 1 | 1 | 4 | 4 | 1 | 1 |

Table 10: The PID-T5 hyperparameters used in the training and testing.

key and value cached tensors from the encoder. The inverse operational intensity of PIE decoder's cross-attention can be denoted as follows,

$$\mathcal{R}^{\text{Dec-cross}}_{\text{PIE}} = \frac{\overbrace{Ub(n_s+n_p)n_td + Ubn_td + n_td^2}^{\text{memory access}}}{\underbrace{Ubn_td^2}_{\text{\# operations}}}$$

$$= \underbrace{\frac{n_s+n_p+1}{d}}_{\text{dominant term}} + \frac{1}{Ub},$$

where the prompts are encoded with the input in the encoder; hence the length of cached tensors is $n_s + n_p$. The dominant term results in a serious bottleneck especially when long input $n_s$ is fed into the model.

Similar to the PID decoder's self-attention, we consider the cross-attention on prompts and output separately. The ratio for encoding prompts is as follows,

$$\mathcal{R}^{\text{Dec-cross,prompt}}_{\text{PID}} = \frac{\overbrace{bn_sd + Ubn_pd + d^2}^{\text{memory access}}}{\underbrace{Ubn_pd^2}_{\text{\# operations}}}$$

$$= \underbrace{\frac{1}{d}\cdot\left(\frac{n_s}{Un_p}+1\right)}_{\text{dominant term}} + \frac{1}{Ubn_p}.$$

In the dominant term, the input length $n_s$ is divided by the factor of $Un_p$.

On the other hand, the ratio of decoding output tokens is

$$\mathcal{R}^{\text{Dec-cross,output}}_{\text{PID}} = \frac{\overbrace{bn_sn_td + Ubn_td + n_td^2}^{\text{memory access}}}{\underbrace{Ubn_td^2}_{\text{\# operations}}}$$

$$= \underbrace{\frac{1}{d}\cdot\left(\frac{n_s}{U}+1\right)}_{\text{dominant term}} + \frac{1}{Ub}.$$

Similarly, in the dominant term, the input length $n_s$ is divided by the factor of $U$. Overall, in PID decoder's cross-attention, the dominant term of the incremental decoding is reduced by a factor of $U$ or $Un_p$ since PID shares the same input key and value cached tensors and broadcast the matrix operations while performing the cross-attention.

## B Training Details

Table 10 presents the hyperparameters selected for the training and testing phases. We executed a search for the optimal learning rate across the following set of values: $\{5\times10^{-4}, 3\times10^{-4}, 1\times10^{-4}, 7\times10^{-5}, 5\times10^{-5}, 3\times10^{-5}\}$ to identify the most effective learning rate for each dataset. All reported values represent the medians of three different random runs. The rest of the hyperparameters are the same as the default values in HuggingFace transformer package. Training time varies because of dataset size, model size, model configuration and training procedure. Our experiments, which utilize $T5_{\text{base}}$ as the primary framework, are carried out using a single NVIDIA A40. Training $T5_{\text{base}}$ and PID-$T5_{\text{base}}$ on the MultiWoZ 2.4 dataset typically requires approximately 5 GPU hours, whereas it takes around 46 GPU hours for PIE-$T5_{\text{base}}$. In the case of ACI-Bench, where the dataset is relatively small, $T5_{\text{base}}$, PIE-$T5_{\text{base}}$, and PID-$T5_{\text{base}}$ require roughly 4 GPU hours each. On the other hand, for RadQA, PIE-$T5_{\text{base}}$ takes 2 hours, while PID-$T5_{\text{base}}$ requires 3 GPU hours. When switching to $T5_{\text{large}}$, the required GPU training time increases by a factor of 2 to 3 times compared to the $T5_{\text{base}}$ models.

We use t5-base[3] and t5-large[4] checkpoints downloaded from HuggingFace as initialization for MultiWoZ 2.4 and ACI-Bench. For RadQA, we follow the previous work (Lehman et al., 2023) to use pretrained clinical T5 models.[5]

## C Datasets

In terms of data preprocessing, we follow the previous works (Ye et al., 2022a; Yim et al., 2023; Lehman et al., 2023) to process MultiWoZ 2.4, ACI-Bench and RadQA, respectively. The dataset statistics are shown in Table 9.

## D License of Artifacts

The licensing for the code from HuggingFace's transformers (Wolf et al., 2020) falls under the Apache License, Version 2.0. PyTorch (Paszke et al., 2019) is open-source software released under the modified BSD license. Meanwhile, the calflops (Ye, 2023) code is protected under the MIT License. Detailed conditions for utilizing our artifacts will be provided within the package we distribute.

[3]https://huggingface.co/google-t5/t5-base

[4]https://huggingface.co/google-t5/t5-large

[5]https://physionet.org/content/clinical-t5/1.0.0/