A DeepSeek-R1 Results

Table 6 presents end-to-end results for DeepSeek-R1 on MultihopRAG and NarrativeQA datasets, evaluated on 16×H20 and 32×H20 GPUs. ContextPilot achieves 1.81× and 1.52× throughput improvements on MultihopRAG and NarrativeQA respectively, with these speedups remaining consistent across both 16 and 32 GPU deployments using context-aware routing.

Table 6. DeepSeek-R1 results formatted for vertical readability. By stacking hardware configurations, the table width is minimized while preserving all data points.

Method	Hardware	Prefill TP (tok/s)	Cache Hit	F1 (%)
	Dataset: MultihopRAG			
Vanilla	16×H20	9636.69	5.12%	64.15
	32×H20	18406.08	4.17%	64.15
ContextPilot w/o Annotations	16×H20	17498.75	60.37%	64.09
	32×H20	33072.64	58.41%	64.09
ContextPilot (Ours)	16×H20	**17498.75**	**60.37%**	**64.68**
	32×H20	**33072.64**	**58.41%**	**64.68**
	Dataset: NarrativeQA			
Vanilla	16×H20	8687.98	6.08%	40.20
	32×H20	16247.32	5.14%	40.20
ContextPilot w/o Annotations	16×H20	13201.52	38.24%	40.38
	32×H20	24686.84	35.71%	40.38
ContextPilot (Ours)	16×H20	**13201.52**	**38.24%**	**41.08**
	32×H20	**24686.84**	**35.71%**	**41.08**

B Attention Map Analysis

Since context engineering (Rajasekaran et al., 2025) and in-context learning (Kirsch et al., 2022) strongly influence model inference, we analyze attention patterns when explicit annotations are introduced to recall the original relevance ranking.

Figure 9 and Figure 10 compare the final-layer attention maps of Qwen3 and LLaMA3.3 under this setup. When given explicit document-priority cues, both models exhibit consistent attention behaviors despite architectural differences. They correctly focus on document tokens ([Doc_1], [Doc_2], [Doc_3]), reflecting awareness of the mismatch between the aligned and original sequences, as indicated by intersections between queries in the annotation region and keys in the context region. As the context re-aligns with the original sequence, both models emphasize ([Doc_2]) while parsing ([Doc_1]) and ([Doc_3]), showing that the cue ([Doc_2] > [Doc_1] > [Doc_3]) effectively directs cross-document attention. Hence, explicit annotations reshape internal attention, aligning it with semantic rather than positional priority.

This finding supports a central hypothesis of ContextPilot: explicit annotations can make a comeback from the accuracy lost to alignment and filtering by re-establishing alignment with the original retrieval semantics.

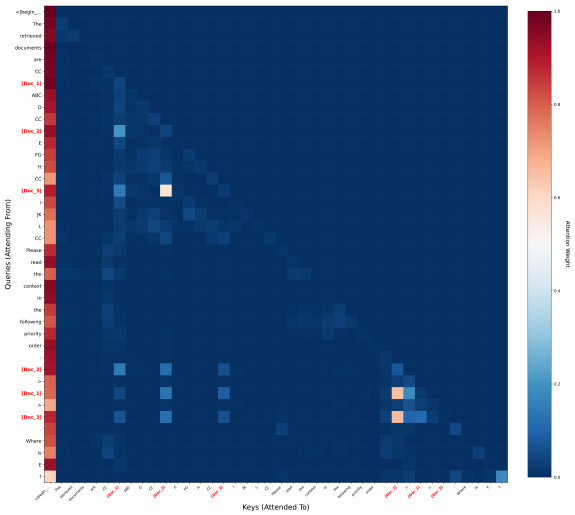

Figure 9. Attention map of the last layer attention of LLaMA3.3 (Head 14) for the prompt "The retrieved documents are [Doc_1] ABCD [Doc_2] EFGH [Doc_3] IJKL. Please read the context in the following priority order: [Doc_2] > [Doc_1] > [Doc_3]. Where is E?".

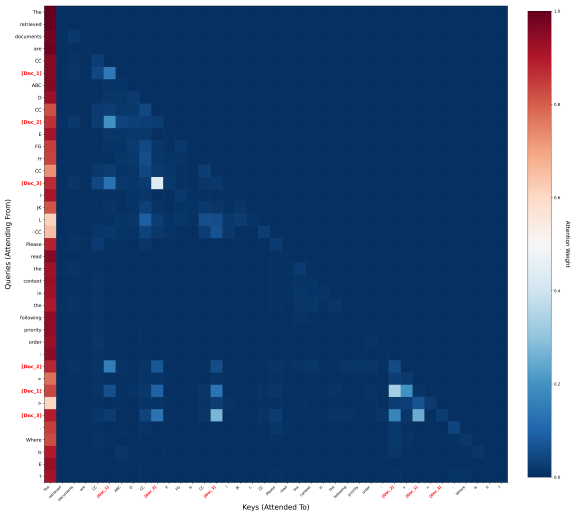

Figure 10. Attention map of the last layer attention of Qwen3 (Head 9) for the prompt "The retrieved documents are [Doc_1] ABCD [Doc_2] EFGH [Doc_3] IJKL. Please read the context in the following priority order: [Doc_2] > [Doc_1] > [Doc_3]. Where is E?".

C Document Access Distribution

Figure 11 shows the cumulative distribution of document access frequency across three RAG datasets. A small fraction of documents accounts for the majority of retrievals: the top

20% most frequently accessed documents cover 79.2% of retrieval events on MultihopRAG, 57.4% on NarrativeQA, and 49.6% on QASPER. This heavy-tailed distribution confirms that real-world RAG workloads exhibit substantial context overlap across sessions, motivating context reuse through prefix-aligned alignment.

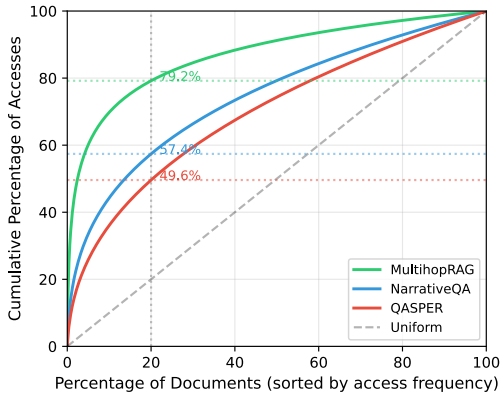

Figure 11. Document access distribution (CDF) across three datasets. The vertical dashed line marks 20% of documents; horizontal lines indicate the cumulative percentage of accesses from these top documents.

D Additional Evaluation Details

D.1 Time-Series Metrics

Figure 12 shows how cache hit ratio evolves as the workload progresses. ContextPilot maintains approximately 34% cache hit ratio compared to baseline's 7% throughout the entire workload, demonstrating a sustained 5× improvement that is not a transient warm-up effect.

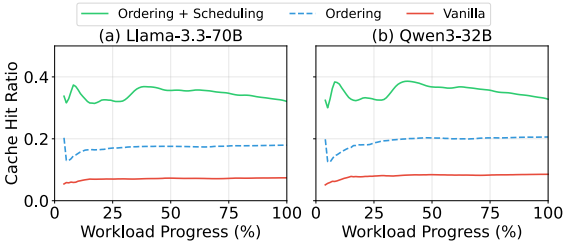

Figure 12. Cache hit ratio over workload progress for Llama-3.3-70B and Qwen3-32B. ContextPilot maintains ∼5× higher hit ratio throughout execution.

Figure 13 presents cumulative cached tokens as a metric for radix tree prefix reuse. ContextPilot achieves 10.33M cached tokens versus baseline's 2.42M at completion on Llama3.3-70B-Instruct (4.27×), and 10.50M versus 2.75M on Qwen3-32B (3.82×). The "w/o Scheduling" variant (6.85M, 2.83×) confirms that both alignment and scheduling contribute to the improvement.

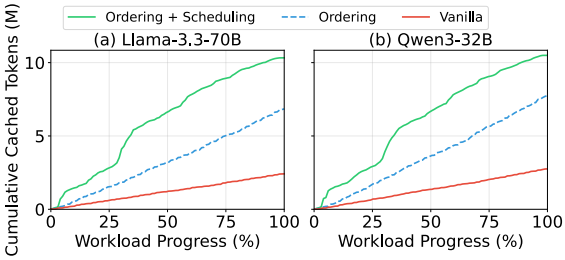

Figure 13. Cumulative cached tokens (radix tree reuse) over workload progress for Llama-3.3-70B and Qwen3-32B. ContextPilot achieves 4× better prefix reuse.

D.2 Accuracy Breakdown

Table 7 provides a detailed breakdown of accuracy contributions from each ContextPilot component.

Table 7. Accuracy breakdown by component. Alignment alone: ≤1% F1 variation; adding annotations: +1.4–4.4% gains over aligned baseline.

Model	Configuration	MultihopRAG	NarrativeQA
Qwen3-32B	Baseline	60.4%	28.4%
	+ Alignment	60.0%	28.2%
	+ Annotation	64.4%	29.6%
	+ Scheduling	**64.4%**	**29.6%**
Qwen3-4B	Baseline	35.2%	16.0%
	+ Alignment	34.5%	15.2%
	+ Annotation	36.6%	17.3%
	+ Scheduling	**36.6%**	**17.3%**

D.3 Per-Request Overhead

Table 8 reports the per-request overhead of ContextPilot components, measured on 2K requests with $k = 15$ on NVIDIA A6000.

Table 8. Per-request overhead breakdown. Total overhead (∼0.7 ms) is negligible compared to seconds of prefill latency.

Component	Latency (ms)
Search	0.068
Alignment	0.047
De-duplication	0.600
Total	**∼0.7**

E OpenClaw Pipeline Diagram

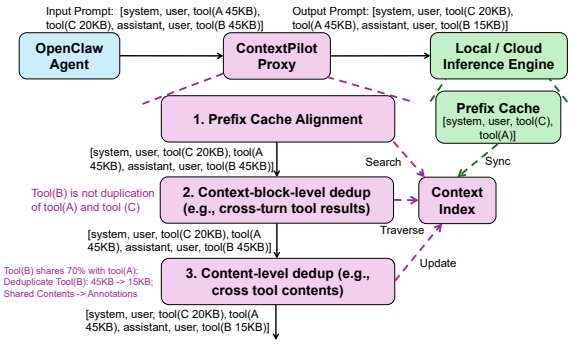

Figure 14. The OpenClaw + ContextPilot pipeline. The OpenClaw agent sends requests to the ContextPilot proxy, which aligns prompts with the prefix cache of the inference engine, deduplicates overlapping context both across turns and within context blocks, and forwards optimized prompts to the inference engine.

F System Overhead with Zero Context Overlap

Zero context overlap represents the worst case for ContextPilot, as it isolates pure system overhead with no reuse benefit. Using a synthetic RAG workload with no retrieval overlap, ContextPilot adds only 0.72 s of prefill latency for 1K contexts (one-hour job), demonstrating that querying the context index during operation incurs negligible overhead.

G Impact of Prefix Cache Size

The prefix KV-cache stores precomputed attention states for reuse across requests; a larger cache retains more contexts, increasing the chance of a cache hit and reducing redundant prefill computation. We evaluate this effect on MultihopRAG across A6000 (48 GB) and H100 (80 GB) GPUs. Because ContextPilot aligns contexts with the prefix cache to maximize overlap, it benefits disproportionately from larger caches: SGLang hit ratios improve from 29.64% to 33.97%, and vLLM from 35.90% to 43.4%, translating directly into higher prefill throughput. In contrast, baselines see smaller gains since their context alignment does not systematically exploit the additional capacity.

H Detailed Algorithm Pseudocode

This appendix collects all algorithm pseudocode referenced in the main text. Algorithm 1 details the context index tree search, Algorithm 2 describes context alignment for prefix sharing, Algorithm 3 formalizes context de-duplication, Algorithm 4 provides the full pseudocode for context index construction via hierarchical clustering, and Algorithm 5 details the tree-based request grouping and scheduling procedure.

Algorithm 1 Context Index Tree Search

Require: Context $C = \{b_1, \ldots, b_k\}$, root node R
Ensure: Search path, best matching node

$cur \leftarrow R$, $path \leftarrow []$
while cur has children **do**
  $best \leftarrow \arg\min_{c \in cur.\text{children},\, C \cap c.\text{blocks} \neq \emptyset} \textsc{Dist}(C, c)$
  if no overlapping child found **then**
    break
  end if
  $path.\textsc{append}(\text{index of } best)$
  if $best$ is leaf **then**
    return $path$, $best$
  end if
  $cur \leftarrow best$
end while
return $path$, cur

Algorithm 2 Context Alignment for Prefix Sharing

Require: Context C with context blocks, Context Tree root node R
Ensure: Aligned context C'

$bestMatch \leftarrow \textsc{FindBestMatchNode}(C, R)$
if $bestMatch =$ null **then**
  $C' \leftarrow C$
  return C'
end if
$prefix \leftarrow bestMatch.context$
$remaining \leftarrow C \setminus prefix$
$C' \leftarrow prefix \cup remaining$
return C'

function $\textsc{FindBestMatchNode}(C, R)$
if C is initialization context **then**
  return $C.parent$
else
  return $\textsc{SearchTree}(C, R)$
end if
end function

Algorithm 3 Context De-duplication

Require: New context C_{new}, context index $\mathcal{I}$, conversation ID id, chunk modulus M
Ensure: Deduplicated context C', location annotations H

$S \leftarrow \mathcal{I}.\text{seen\_blocks}[id]$ ▷ blocks indexed in prior turns
$V \leftarrow \mathcal{I}.\text{seen\_subblocks}[id]$ ▷ sub-block hashes from prior turns
$C' \leftarrow [], H \leftarrow []$
for each block b in C_{new} **do**
  if $b \in S$ **then**
    $h \leftarrow$ "refer to $[b]$ in previous conversation"
    $C'.\text{APPEND}(h)$; $H.\text{APPEND}(h)$
  else
    $\{s_1, \ldots, s_m\} \leftarrow \text{CDC}(b, M)$ ▷ content-defined chunking
    for $j = 1$ to m **do**
      $f \leftarrow \text{HASH}(s_j)$
      if $f \in V$ **and** $V[f] \neq b$ **then**
        Replace s_j with location annotation to $V[f]$
      else
        $V[f] \leftarrow b$
      end if
    end for
    $C'.\text{APPEND}(\text{CONCAT}(s_1, \ldots, s_m))$
  end if
end for
$S \leftarrow S \cup C_{\text{new}}$ ▷ register for future turns
$\mathcal{I}.\text{seen\_subblocks}[id] \leftarrow V$ ▷ persist sub-block hashes
return C', H

Algorithm 4 Context Index Construction via Hierarchical Clustering

Input: Batch of n contexts $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, distance function $d(\cdot, \cdot)$

// Phase 1: Distance computation and clustering
$D \leftarrow$ pairwise distances using $d(s_i, s_j)$ ▷ GPU or CPU
$Z \leftarrow \text{LINKAGE}(D)$ ▷ hierarchical clustering
// Phase 2: Build tree with deduplication
for $i = 1$ to n **do**
  Create leaf node v_i with v_i.content $\leftarrow s_i$
  if $\exists\, v_j$ with v_j.content $= s_i$ **then**
    Redirect $v_i \rightarrow v_j$; v_j.freq $+= 1$ ▷ dedup
  end if
end for
for each merge (c_1, c_2, δ) in Z **do**
  $v \leftarrow$ new internal node
  v.content $\leftarrow c_1$.content $\cap\, c_2$.content
  v.children $\leftarrow [c_1, c_2]$; c_1.parent, c_2.parent $\leftarrow v$
end for
Remove empty internal nodes; relink children to grandparents
// Phase 3: Top-down prefix alignment
Compute search paths from root to all nodes
for each node v in BFS order from root **do**
  if v is not root **and** v.parent.docs $\neq \emptyset$ **then**
    v.docs $\leftarrow v$.parent.docs $\oplus (v.\text{docs} \setminus v.\text{parent.docs})$
  end if
end for
return aligned contexts from leaf nodes, search paths

Algorithm 5 Search-Path-Based Request Grouping and Scheduling

Input: N contexts with search paths $\{P_1, \ldots, P_N\}$ from alignment
Output: Scheduled execution order
// Phase 1: Group by root prefix ▷ $O(N)$
$\mathcal{G} \leftarrow$ empty map
for $i = 1$ to N **do**
  $key \leftarrow P_i[0]$ ▷ first element of search path
  $\mathcal{G}[key].\text{APPEND}(i)$
end for
// Phase 2: Sort within each group ▷ $O(N \log N)$
for each group $G \in \mathcal{G}$ **do**
  Sort G by $|P_i|$ descending ▷ longer paths first
end for
// Phase 3: Order groups and flatten
Sort groups by $|G|$ descending ▷ largest groups first
return concatenation of all groups