Appendix

A. *Results on Latest Close-Source Model*

To further demonstrate the effectiveness and generalizability of LongCodeZip, we conduct experiments on the latest close-sourced model, Claude 3.7 Sonnet. For the compression stage, we use Qwen2.5-Coder-7B as the compression model. The following tables present the results across three different tasks. We can observe that LongCodeZip still achieves significant performance improvements over other compression baselines on this powerful close-sourced model, revealing its effectiveness and generalizability.

Table VIII presents the results on the Long Code Completion task with Claude 3.7 Sonnet. Our approach demonstrates strong performance, achieving an ES score of 66.27 and an EM score of 40.29, which is on par with the no-compression baseline while operating at a 4.3x compression ratio. Notably, LongCodeZip surpasses the best-performing baseline, RAG (Function Chunking), which scored 63.55 in ES at a less efficient 3.1x ratio. This result underscores the effectiveness of our method in preserving critical information for code completion with significantly greater compression efficiency, even on state-of-the-art closed-source models.

TABLE VIII: Results on Long Code Completion with Claude 3.7 Sonnet

Model	**Method**	**ES**	**EM**	**Ratio**
Claude 3.7 Sonnet	*No Compression*	66.24	41.21	1.0x
	No Context	43.97	14.28	-
	Random Token	47.61	14.07	4.4x
	Random Line	52.61	22.26	4.5x
	RAG (Sliding Window)	61.44	34.02	2.8x
	RAG (Function Chunking)	63.55	36.75	3.1x
	LLMLingua	46.58	15.12	3.4x
	LLMLingua-2	49.02	16.17	4.4x
	LongLLMLingua	57.58	27.72	3.2x
	DietCode	54.00	19.74	3.4x
	SlimCode	53.03	20.79	4.5x
	LongCodeZip	**66.27**	**40.29**	4.3x

The results for the Long Module Summarization task are shown in Table IX. LongCodeZip remains the most competitive method, achieving a *CompScore* of 61.47, slightly outperforming the no-compression baseline (60.72) at a 1.7x compression ratio. In contrast to other baselines like RAG (58.03) and LLMLingua-2 (57.85), our approach demonstrates a clear advantage. This highlights the effectiveness of our method in preserving the most relevant semantic content for summarization, indicating that removing distracting information can even improve performance on complex summarization tasks.

TABLE IX: Results on Long Module Summarization with Claude 3.7 Sonnet

Model	**Method**	**CompScore**	**Ratio**
Claude 3.7 Sonnet	*No Compression*	60.72	1.0x
	No Context	6.58	-
	Random Token	37.45	1.8x
	Random Line	50.12	1.8x
	RAG (Sliding Window)	58.03	1.7x
	RAG (Function Chunking)	44.56	2.1x
	LLMLingua	43.21	1.7x
	LongLLMLingua	50.86	1.5x
	LLMLingua-2	57.85	2.1x
	DietCode	38.82	2.1x
	SlimCode	48.11	2.2x
	LongCodeZip	**61.47**	1.7x

Table X details the results on the multilingual RepoQA task. Our approach consistently achieves the best performance, with an average accuracy of 90.7% that surpasses even the no-compression baseline (89.5%) while compressing the context by 4.5x. The performance gain is particularly pronounced when compared to other compression methods; for instance, LongCodeZip outperforms the next-best baseline, LongLLMLingua (74.2%), by a margin of over 16 percentage points. This superior performance across all evaluated languages underscores our method's effectiveness in retaining precise, structured code information necessary for retrieval-based QA.

TABLE X: Results on RepoQA with Claude 3.7 Sonnet.

Method	**Py**	**C++**	**Java**	**TS**	**Rust**	**Go**	**Avg.**	**Ratio**
Claude 3.7 Sonnet								
No Compression	87.4	80.1	92.6	96.7	86.3	93.6	89.5	1.0x
No Context	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-
Random Token	3.6	3.1	4.2	2.1	4.2	7.3	4.1	3.6x
Random Line	6.2	11.4	22.9	10.4	9.4	13.5	12.3	3.5x
RAG (Sliding Window)	66.6	67.6	70.7	74.9	59.3	82.2	70.2	3.7x
RAG (Function Chunking)	56.2	48.9	61.4	40.6	60.3	71.8	56.5	4.3x
LLMLingua	5.2	7.3	9.4	11.4	4.2	16.6	9.0	4.1x
LLMLingua-2	1.0	2.1	8.3	1.0	1.0	4.2	2.9	4.6x
LongLLMLingua	72.8	65.5	73.8	70.7	81.1	81.1	74.2	4.3x
DietCode	17.7	-	36.4	-	-	-	27.1	3.7x
SlimCode	20.8	-	49.9	-	-	-	35.4	4.3x
LongCodeZip	**95.7**	**81.1**	**90.5**	**88.4**	**89.4**	**98.8**	**90.7**	4.5x