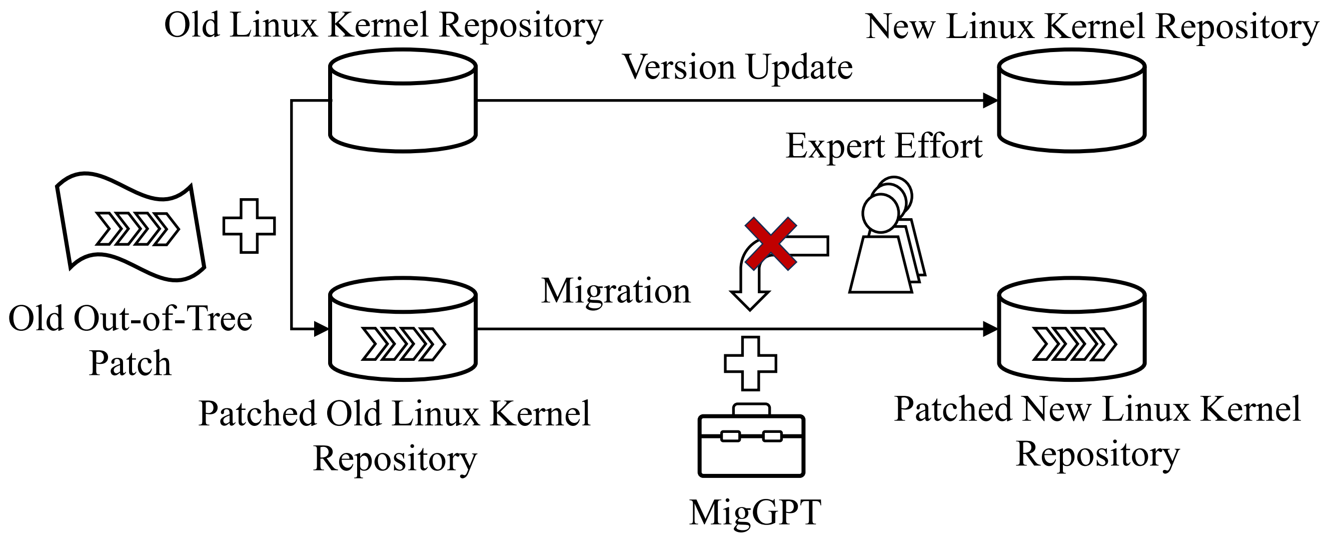
MigGPT: Harnessing Large Language Models for Automated Migration of Out-of-Tree Linux Kernel Patches Across Versions

Pucheng Dang[1,2,3] **Di Huang**[1] **Dong Li**[1,2,3 *] **Kang Chen**[4]
Yuanbo Wen[1] **Qi Guo**[1] **Xing Hu**[1,3]
[1] SKLP, Institute of Computing Technology, CAS
[2] University of Chinese Academy of Sciences
[3] Zhongguancun Laboratory
[4] Tsinghua University

Out-of-tree Patch Migration Across Linux Kernel Versions

The Linux kernel is a widely used open-source operating system whose flexibility allows developers to extend functionality or adapt hardware through out-of-tree patchesc. These patches are developed independently from the mainline kernel but require continuous maintenance as the kernel evolves. Currently, this process relies on manual effort, taking several weeks and requiring skilled engineers, resulting in high costs.


Old Linux Kernel Repository
New Linux Kernel Repository
Version Update
Expert Effort
Old Out-of-Tree Patch
Migration
Patched Old Linux Kernel Repository
Patched New Linux Kernel Repository
MigGPT

Contributions:

- We have developed a robust migration benchmark, encompassing three real-world projects. To the best of our knowledge, this is the first benchmark for out-of-tree kernel patch migration that can assess performance across diverse migration tools.
- We propose CFP, a carefully designed data structure that encapsulates the structural and critical information of code snippets, providing essential migration context for LLMs. Based on this, we introduce MigGPT, a framework to assist humans in automating out-of-tree kernel patch migration and maintenance.
- We conduct comprehensive experiments on both closed-source models and open-source models. The results demonstrate that MigGPT achieved an average migration accuracy of 74.07% (↑45.92%), representing a significant improvement over vanilla LLMs.

Related Works

Code migration techniques have primarily focused on:

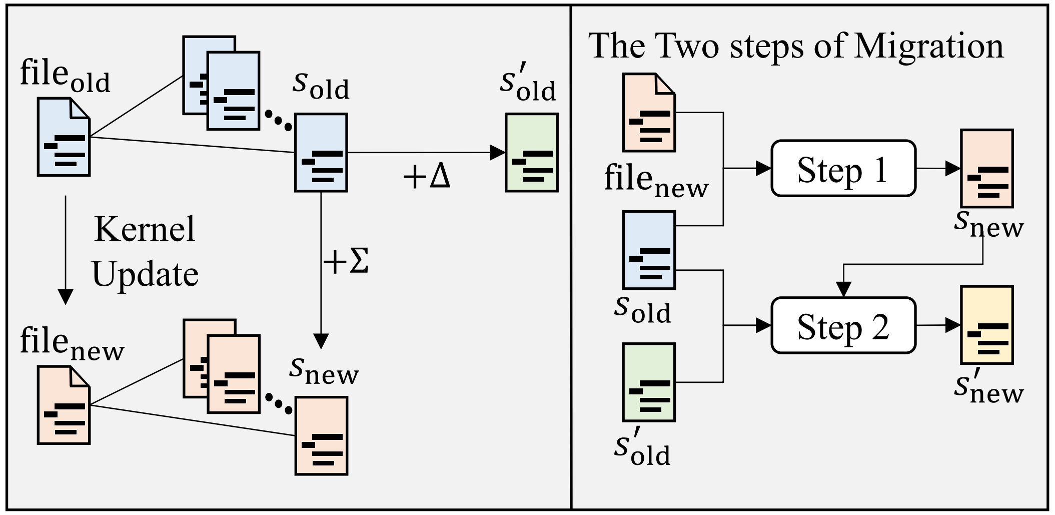
- API migration across versions (Xing & Stroulia, 2007; Lamothe et al., 2022; Fazzini et al., 2019; Haryono et al., 2020; Thung et al., 2019; Ketkar et al., 2019; Rolim et al., 2017; Dilhara et al., 2023)
- CVE backporting (Shi et al., 2022; Shariffdeen et al., 2021a,b; Yang et al., 2023)

Traditional code migration techniques, such as static analysis, support only partial scenarios (e.g., API version updates, CVE patch backporting). However, they rely on predefined rules and are unable to handle complex changes such as namespace modifications or control-flow dependencies.

Benchmark

We have built a robust migration testing benchmark using out-of-tree kernel patches from real-world projects, specifically focusing on three open-source initiatives: RT-PREEMPT, Raspberry Pi Linux, and HAOC. Considering the states of Δ and Σ we can categorize the migration types into two classes:

Class	Type 1	Type 2
Number	80 (59.3%)	55 (40.7%)
Formalization	$\Delta \neq \emptyset, \Sigma \neq \emptyset,$ $\forall \delta \in \Delta, \forall \sigma \in \Sigma, \langle \delta, \sigma \rangle = 0$	$\Delta \neq \emptyset, \Sigma \neq \emptyset,$ $\forall \delta \in \Delta, \forall \sigma \in \Sigma, \langle \delta, \sigma \rangle \neq 0$
Description	Their changes do not overlap or conflict with each other.	Their modifications overlap on the same lines of code, leading to conflicts.



Challenge

Challenges of using vanilla LLMs:

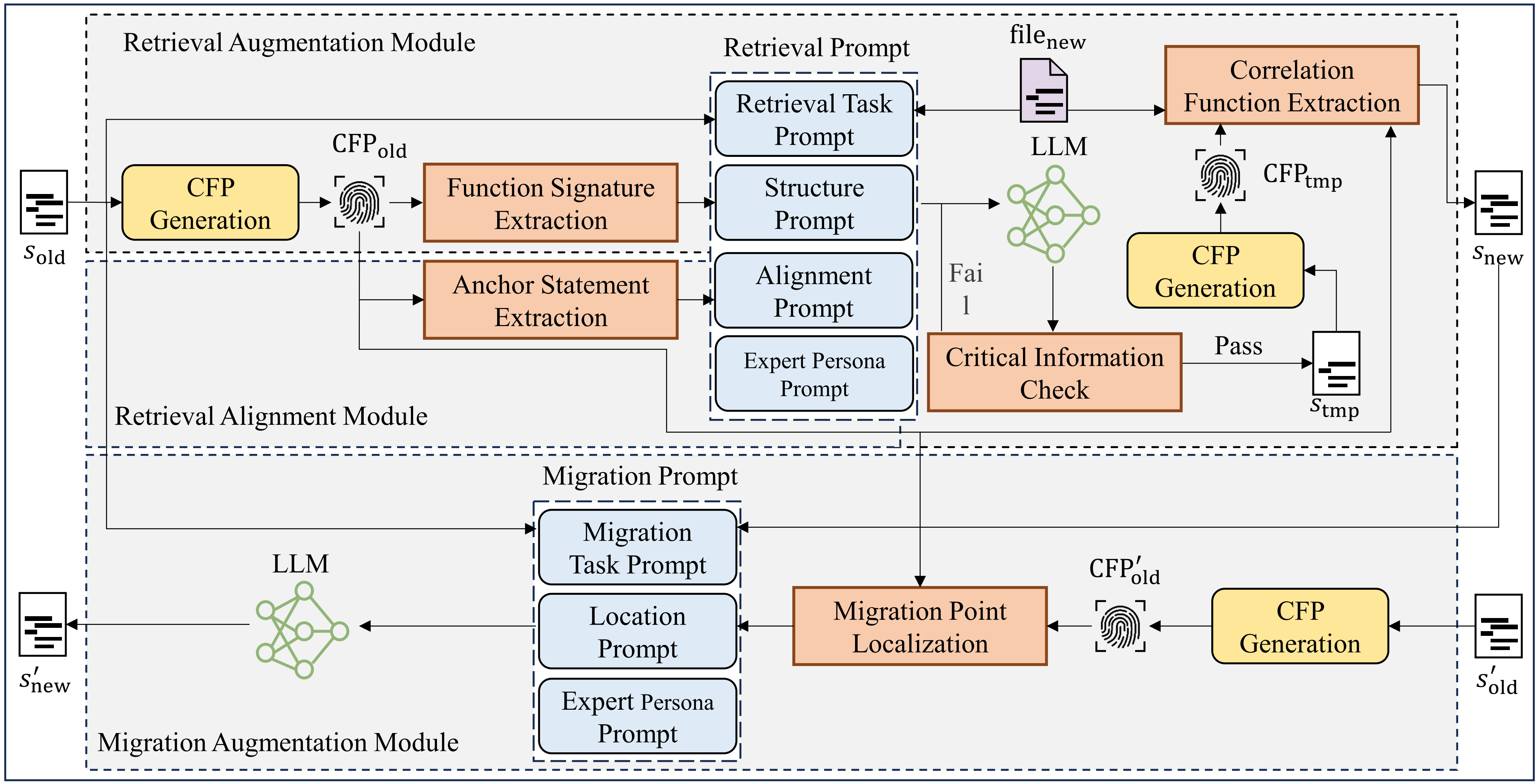
- **Challenge 1 (Structural Ambiguity):** When identifying the code snippet s_{new} in $file_{new}$ for migration, retrieval errors can occur. LLMs often struggle to locate function definitions within s_{new} due to interference from similar function structures, leading to inaccuracies that affect subsequent migration stages.

- **Challenge 2 (Boundary Indeterminacy):** This challenge occurs when retrieving s_{new} from $file_{new}$. Due to the inherent randomness in LLM-generated responses, discrepancies often arise between the start and end lines of s_{new} identified by the LLM and those retrieved by human developers. This indeterminacy can result in missing or extraneous lines, significantly compromising migration outcomes where precise code segment boundaries are critical.

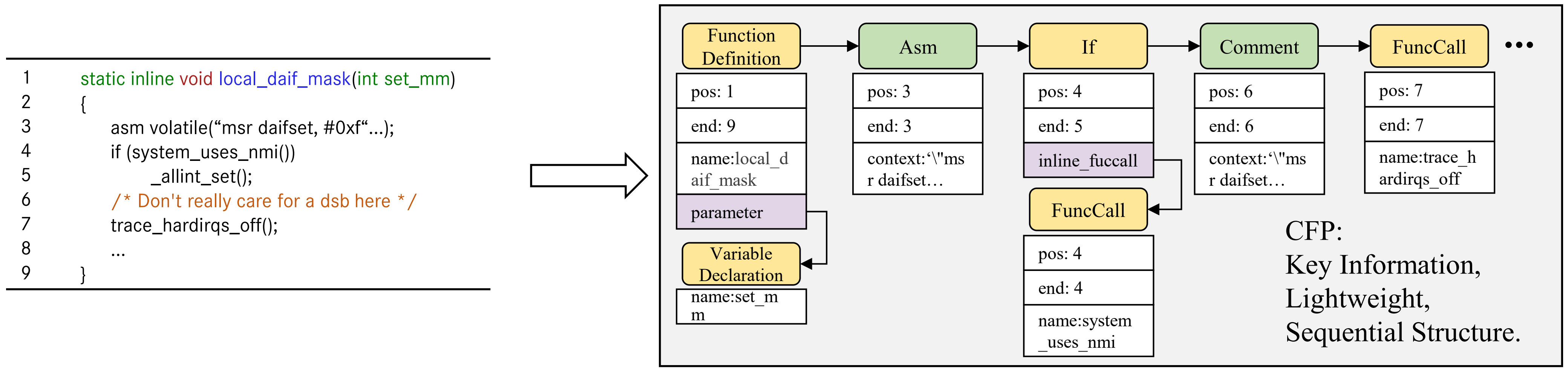
- **Challenge 3 (Missing Associated Fragments):** This challenge occurs when retrieving s_{new} from $file_{new}$. During Linux kernel upgrades, code blocks from older versions may be split into fragments in the new version for standardization or reuse. LLMs often fail to identify and retrieve all these fragments, leading to incomplete s_{new}. This results in errors during out-of-tree kernel patch migration due to missing code segments.

- **Challenge 4 (Ambiguous Migration Points):** This challenge arises during the migration of s_{new} to s'_{new}. Although the information provided by s_{old} and s'_{old} is sufficient to accurately infer the migration point, LLMs frequently fail to precisely identify these points. This ambiguity results in errors when determining the correct location for migration.

Framework


Retrieval Augmentation Module
Retrieval Prompt
$file_{new}$
Correlation Function Extraction
Retrieval Task Prompt
CFP_{old}
LLM
CFP_{tmp}
s_{old}
CFP Generation
Function Signature Extraction
Structure Prompt
CFP Generation
s_{new}
Fai
l
Anchor Statement Extraction
Alignment Prompt
Pass
Critical Information Check
Expert Persona Prompt
s_{tmp}
Retrieval Alignment Module
Migration Prompt
Migration Task Prompt
LLM
CFP'_{old}
Migration Point Localization
CFP Generation
Location Prompt
s'_{new}
s'_{old}
Expert Persona Prompt
Migration Augmentation Module

- **Code Fingerprint (CFP):** CFP records both the content and positional information for each line of statements, encompassing all C language statements, including comments and inline assembly. CFP preserves critical information such as comments and inline assembly, which is essential for migrating out-of-tree kernel patches.



```
1    static inline void local_daif_mask(int set_mm)
2    {
3        asm volatile(“msr daifset, #0xf“...);
4        if (system_uses_nmi())
5               _allint_set();
6        /* Don't really care for a dsb here */
7        trace_hardirqs_off();
8        ...
9    }
```

- **Retrieval Augmentation Module (address challenge 1 and 3):** the retrieval augmentation module achieves this by constructing a CFP_{old} structure for s_{old}. By analyzing CFP_{old}, the module extracts the function signatures of the function definitions contained within s_{old}. These function signatures are then used to build a prompt to describe the structure information (“Structure Prompt”), which is incorporated into the input fed to the LLM.
- **Retrieval Alignment Module (address challenge 2):** we utilize the code fingerprint structure CFP_{old} of s_{old} to obtain the CFP statements for the first and last lines. These statements are used to construct an “Alignment Prompt”, which describes the information of the first and last lines and is included as part of the input to the LLM.
- **Migration Augmentation Module (address challenge 4):** we leverage information from the old version code snippet s_{old} and its modified counterpart s'_{old} to determine the number and location of modifications made to the out-of-tree kernel patch. This information is used to construct a “Location Prompt’” that assists the LLM in accurately identifying the number and location of migration points.

Evaluation

Table 1: The accuracy of the MigGPT-augmented LLMs compared to vanilla LLMs on retrieving target code snippets.

LLM	Method	Type 1 (80)			Type 2 (55)			All (135)		
		Best Match	Semantic Match	Human Match	Best Match	Semantic Match	Human Match	Best Match	Semantic Match	Human Match
GPT-3.5	Vanilla	20.00%	33.75%	26.25%	20.00%	25.45%	27.27%	20.00%	30.37%	26.67%
	MIGGPT	68.75%	68.75%	71.25%	61.82%	54.55%	70.91%	65.93%	62.96%	71.11%
GPT-4-turbo	Vanilla	60.00%	67.50%	65.00%	69.09%	76.36%	78.18%	63.70%	71.11%	70.37%
	MIGGPT	91.25%	95.00%	96.25%	81.82%	83.64%	89.09%	87.41%	90.37%	93.33%
OpenAI-o1	Vanilla	76.25%	82.50%	80.00%	81.82%	85.45%	92.73%	78.52%	83.70%	85.19%
	MIGGPT	96.25%	97.50%	96.25%	85.45%	89.09%	92.73%	91.85%	94.07%	94.81%
DeepSeek-V3	Vanilla	68.75%	71.25%	72.50%	74.55%	78.18%	78.18%	71.11%	74.07%	74.81%
	MIGGPT	92.50%	93.75%	95.00%	85.45%	78.18%	89.09%	89.63%	87.41%	92.59%
DeepSeek-R1	Vanilla	72.50%	76.25%	77.50%	63.64%	70.91%	74.55%	68.89%	74.07%	76.30%
	MIGGPT	95.00%	95.00%	95.00%	80.00%	85.45%	87.27%	88.89%	91.11%	91.85%
Llama-3.1-8B	Vanilla	37.50%	46.25%	43.75%	43.64%	47.27%	45.45%	40.00%	46.67%	44.44%
	MIGGPT	77.50%	80.00%	81.25%	70.91%	74.55%	78.18%	74.81%	77.78%	80.00%
Llama-3.1-70B	Vanilla	58.75%	65.00%	63.75%	61.82%	72.73%	75.55%	60.00%	68.15%	68.15%
	MIGGPT	91.25%	92.50%	93.75%	80.00%	81.82%	81.82%	86.67%	88.15%	88.89%
Average	Vanilla	56.25%	63.26%	61.25%	59.22%	65.19%	67.42%	57.46%	64.02%	63.70%
	MIGGPT	87.50%	88.93%	76.25%	77.92%	78.18%	84.16%	83.60%	84.55%	87.51%
	↑	+31.25%	+25.67%	+15.00%	+18.70%	+12.99%	+16.74%	+26.14%	+20.53%	+23.81%

- **MigGPT demonstrates exceptional capability in retrieving target code snippets** (Table 1). When paired with a high-performance LLM like GPT-4-turbo, MigGPT achieves a human matching precision of 96.25% for Type 1 samples, significantly outperforming vanilla GPT-4-turbo (65.00%).

Table 2: The accuracy of the MigGPT-augmented LLMs compared to vanilla LLMs on the migration task of target code snippets.

LLM	Method	Type 1 (80)			Type 2 (55)			All (135)		
		Best Match	Semantic Match	Human Match	Best Match	Semantic Match	Human Match	Best Match	Semantic Match	Human Match
GPT-3.5	Vanilla	7.50%	5.00%	8.75%	3.64%	3.64%	5.45%	5.93%	4.44%	7.41%
	MIGGPT	37.50%	46.26%	47.50%	38.18%	41.82%	61.82%	37.78%	44.44%	53.33%
GPT-4-turbo	Vanilla	15.00%	12.50%	18.75%	10.91%	30.91%	23.64%	13.33%	20.00%	20.74%
	MIGGPT	68.75%	82.50%	85.00%	54.55%	76.36%	69.09%	62.96%	80.00%	78.52%
OpenAI-o1	Vanilla	20.00%	28.75%	30.00%	18.18%	30.91%	27.27%	19.26%	29.63%	28.89%
	MIGGPT	77.50%	90.00%	90.00%	60.00%	76.36%	74.55%	70.37%	84.44%	83.70%
DeepSeek-V3	Vanilla	23.75%	37.50%	32.50%	34.55%	54.55%	49.09%	28.15%	44.44%	39.26%
	MIGGPT	81.25%	88.75%	87.50%	65.45%	78.18%	74.55%	74.81%	84.44%	82.22%
DeepSeek-R1	Vanilla	53.75%	60.00%	62.50%	40.00%	54.55%	56.36%	48.15%	57.78%	60.00%
	MIGGPT	72.50%	85.00%	81.25%	69.09%	81.82%	78.18%	71.11%	83.70%	80.00%
Llama-3.1-8B	Vanilla	5.00%	12.50%	16.25%	0%	20.00%	25.45%	2.96%	15.56%	20.00%
	MIGGPT	36.25%	65.00%	67.50%	25.45%	52.73%	56.36%	31.85%	60.00%	62.96%
Llama-3.1-70B	Vanilla	3.75%	16.25%	18.75%	7.27%	27.27%	23.64%	5.19%	20.74%	20.74%
	MIGGPT	62.50%	80.00%	81.25%	47.27%	67.27%	72.73%	56.30%	74.81%	77.78%
Average	Vanilla	18.39%	24.64%	26.79%	16.36%	31.69%	30.13%	17.57%	27.51%	28.15%
	MIGGPT	62.32%	76.78%	77.14%	51.43%	67.79%	69.61%	57.88%	73.12%	74.07%
	↑	+43.93%	+52.14%	+50.36%	+35.06%	+36.10%	+39.48%	+40.32%	+45.61%	+45.92%

- **MigGPT demonstrates outstanding performance in generating migrated code snippets** (Table 2). MigGPT outperforms vanilla LLMs, achieving a 73.12% higher average migration semantic matching precision, a 45.61% relative improvement.

Ablation Study

Table 3: The time cost of MigGPT compared to human experts.

Method	Expert A	Expert B	Expert C	MIGGPT	
				GPT-4-turbo	DeepSeek-V3
Time (days)	14.15	10.89	12.51	0.25	0.27

- **MigGPT is more time-efficient.** We compared MigGPT with three human experts on our out-of-tree patch migration benchmark. As shown in Table 3, MigGPT required only 2.08% of the average time taken by human experts, demonstrating its superior time efficiency.

Table 4: The line edit distance between the failure cases of MigGPT-augmented GPT-4-Turbo and the manual migration results. "$3 \leq dis < 6$" denotes a line edit distance of at least 3 but less than 6.

LLM	Type	$dis < 3$	$3 \leq dis < 6$	$6 \leq dis < 9$	$9 \leq dis$	All
GPT-3.5	Type 1	13	9	8	12	42
	Type 2	8	4	2	7	21
GPT-4-turbo	Type 1	5	1	3	3	12
	Type 2	9	1	0	7	17
DeepSeek-V2.5	Type 1	10	2	3	1	16
	Type 2	8	1	3	4	16
DeepSeek-V3	Type 1	3	2	3	2	10
	Type 2	5	4	1	4	14

- **MigGPT's failed cases only require minor modifications to be corrected.** We evaluate MigGPT's robustness by analyzing failed migration cases (not human-matched) across various samples, measuring line edit distances (insertions, deletions, modifications) between MigGPT's incorrect outputs and human-corrected results. As shown in Table 4, 41% of MigGPT's errors require fewer than three lines of modification to align with correct results, demonstrating its potential to aid in out-of-tree kernel patch migration.

NEURAL INFORMATION PROCESSING SYSTEMS

Thanks