PEARL: Peer-Enhanced Adaptive Radio via On-Device LLM

Ju-Hyung Lee[*1] **Yanqing Lu**[*1,2] **Klaus Doppler**[1]
[1]Nokia Technologies, Sunnyvale, CA, USA
[2]Department of Computer Science, University of Southern California, Los Angeles, CA, USA

Abstract

We present **PEARL** (*Peer-Enhanced Adaptive Radio via On-Device LLM*), a framework for cooperative cross-layer optimization in device-to-device (D2D) communication. Building on our previous work on single-device on-device LLMs [Lee et al., 2025], PEARL extends the paradigm by leveraging both publisher and subscriber states to guide `Wi-Fi Aware` (WA) parameter selection. A context-aware reward, which normalizes latency by application tolerances and modulates energy by device battery states, provides richer supervision for KL-based fine-tuning. We study two lightweight variants: PEARL (Head + Low-Rank Adaptation (LoRA) [Hu et al., 2022]) achieves the best overall performance, while PEARL-Lite (Head-only) delivers sub-20 ms inference at near-identical objective scores. Across synthetic scenarios grounded in real measurements, PEARL improves objective scores over heuristic and compact model baselines and reduces energy by up to 16% in cooperative low-battery cases. These results demonstrate that peer-aware context, reward-aligned training, and head-based efficiency make LLMs practical for always-on, on-device cross-layer control. Code, real-world demo, and dataset are available at `https://github.com/abman23/pearl`.

1 Introduction

Large language models (LLMs) are increasingly being adapted for resource-constrained edge devices [Qu et al., 2025], where they can enable adaptive networking, sensing, and cross-layer optimization [Shao et al., 2024] without relying on the cloud. Our previous work demonstrated that on-device LLMs can improve wireless performance by tuning cross-layer parameters in single-device settings, leveraging local context such as user location and time of day [Lee et al., 2025]. While promising, this prior line of research leaves open the question of how LLM-based optimization can extend beyond isolated devices to cooperative, device-to-device (D2D) scenarios.

In D2D communication, link performance depends not only on the local device state but also on peer conditions such as battery level. For example, when the publisher has ample energy but the subscriber is low on battery, an energy-preserving configuration may be preferable even at the cost of slightly higher latency. Designing a learning system that accounts for such asymmetric, two-sided context remains an open challenge. Moreover, efficient on-device deployment requires parameter-efficient fine-tuning (PEFT) strategies [Bucher and Martini, 2024, Han et al., 2024], but it is unclear how different PEFT modules compare in this setting, or how to ensure that training signals faithfully reflect heterogeneous application and device constraints.

Contributions. We propose **PEARL** (*Peer-Enhanced Adaptive Radio via On-Device LLM*), the first on-device LLM framework for cooperative cross-layer D2D optimization. Our key contributions are:

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: AI and ML for Next-Generation Wireless Communications and Networking (AI4NextG).

[*]Equal contribution. {juhyung.lee, yanqing.lu}@nokia.com

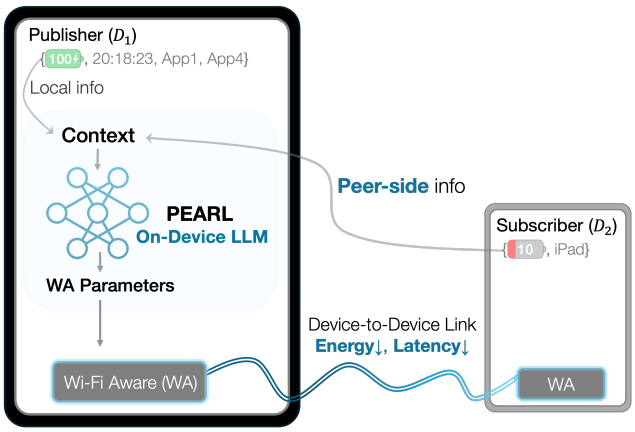


Figure 1: System overview of **PEARL**. The agent runs on the publisher (D_1), combining local context (*e.g.*, battery level, time, running applications) with peer-side context shared by the subscriber (D_2, *e.g.*, battery level, device type). Based on these inputs, PEARL selects WA parameters (`PerformanceMode`, `AccessCategory`), which configure the WA stack to minimize latency and energy on the D2D link.

(i) **Peer-context integration** (Sec. 3.1, 4.3): subscriber-side information is incorporated into the model input, enabling two-sided adaptation beyond single-device optimization.

(ii) **Context-aware reward** (Sec. 3.2, 4.4): a reward function that normalizes latency by application-specific tolerances and modulates energy by device battery state, providing richer supervision for KL-based fine-tuning.

(iii) **Head-based PEFT** (Sec. 3.3, 4.6): lightweight classification heads (PEARL and PEARL-Lite) that achieve strong objective scores with sub-20 ms inference, narrowing the gap to the oracle while remaining deployable on mobile hardware.

Extensive evaluation on `Wi-Fi Aware` (Sec. 4 and Table 3) shows that PEARL consistently outperforms heuristic baselines and compact non-LLM models, demonstrating that peer-aware, reward-aligned training substantially improves both efficiency and robustness in D2D optimization.

2 Problem Formulation

Scenario. We consider a D2D link established via `Wi-Fi Aware` (WA) between a *publisher* D_1 and a *subscriber* D_2 [Camps-Mur et al., 2015]. The publisher transmits application data while selecting cross-layer WA parameters that directly affect link performance. The decision model leverages both local and peer-side context, including device battery levels, device type (*e.g.*, iPhone, iPad), time of day, and foreground application type sampled from a synthetic, time-varying distribution.

Action Space. At each decision step, the model chooses one of 8 discrete parameter tuples (`PerformanceMode`, `AccessCategory`), where `PerformanceMode` $\in$ {`realtime`, `bulk`} and `AccessCategory` $\in$ {`bestEffort`, `background`, `interactiveVideo`, `interactiveVoice`}.

Objective. The objective is to balance link latency and device energy consumption while satisfying application-specific requirements. Both local (publisher) and peer (subscriber) context guide parameter selection. For example, when the subscriber has low battery, the model should prefer energy-preserving modes even at the cost of slightly higher latency.

3 Method

3.1 System Architecture

PEARL builds on a pre-trained large language model (LLM) as a general-purpose feature encoder, and adapts it into a lightweight classifier for cross-layer parameter selection [Wu et al., 2024]. Context features from both publisher and subscriber devices (*e.g.*, battery levels, time, application) are embedded into a structured prompt and passed to the frozen LLM backbone. A

task-specific classification head is attached on top, predicting one of the discrete parameter tuples (`PerformanceMode`, `AccessCategory`).

We consider two variants: PEARL (Head + Low-Rank Adaptation (LoRA) [Hu et al., 2022]), where the classification head is trained jointly with lightweight LoRA adapters inserted into the LLM, and PEARL-Lite (Head-only), where only the head is trained while the backbone remains frozen. For comparison, we also evaluate *LoRA-only*, where decisions are generated through the language modeling head without a classification head.

At inference, context features are encoded into a structured prompt, passed through the frozen LLM encoder, and mapped by the classification head to one of the 8 WA parameter tuples (Fig. 2).

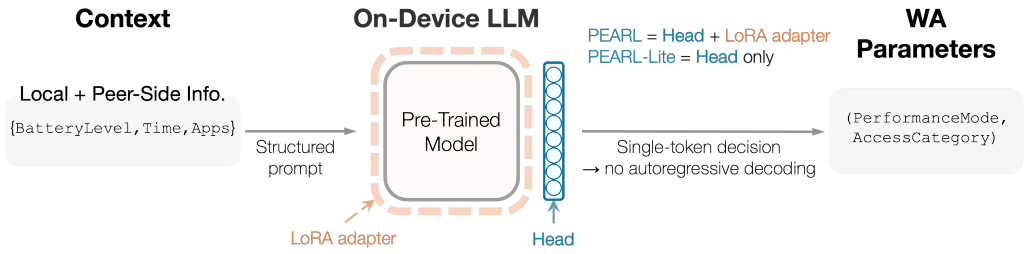


Figure 2: Architecture of PEARL. Context features are encoded into a structured prompt and passed through a frozen LLM encoder. In **PEARL-Lite**, a classification head directly predicts one of 8 WA parameter tuples. In **PEARL**, LoRA adapters are added to the encoder and trained jointly with the head. In both cases, the head produces a single-token decision, avoiding autoregressive decoding.

3.2 Context-Aware Reward Design

We design a reward function that captures both application-specific latency requirements and device battery states. Each is normalized into a *score*, where higher values indicate better performance.

For an application a with latency tolerance L_a and parameters p yielding average latency $\ell(p)$, the latency score is

$$R_{\text{lat}}(a,p) = \max\left(100 - 100 \cdot \tfrac{\ell(p)}{L_a},\, 0\right).$$

For a device d with battery level b_d and energy usage $E(p)$ under parameters p, the energy score is

$$R_{\text{eng}}(d,p) = \frac{b_d}{E(p)}.$$

The overall reward combines these terms across active applications $\mathcal{A}$ and devices $\mathcal{D}$:

$$R(p) = w_L \cdot \tfrac{1}{|\mathcal{A}|} \sum_{a\in\mathcal{A}} R_{\text{lat}}(a,p) - w_P \cdot \tfrac{1}{|\mathcal{D}|} \sum_{d\in\mathcal{D}} \tfrac{1}{R_{\text{eng}}(d,p)}.$$

Here, $\ell(p)$ and $E(p)$ are the measured latency and energy under parameters p, $\mathcal{A}$ is the set of active applications, and $\mathcal{D} = \{\text{publisher}, \text{subscriber}\}$ the two devices considered. Weights w_L and w_P control the trade-off between latency and energy.

This formulation produces a reward distribution that yields *soft labels* for KL-based fine-tuning, preserving relative action preferences and improving generalization (Appendix C, H).

3.3 Post-Training Strategies

We evaluate several post-training strategies for adapting the LLM encoder and prediction head:

- **Supervised Fine-Tuning (SFT):** We compare cross-entropy (CE) loss with Kullback–Leibler (KL) divergence loss. CE uses hard argmax labels, while KL leverages the full reward distribution as soft labels, preserving relative preferences across actions [Hinton et al., 2015].

- **Preference Optimization (DPO):** Optionally, we apply Direct Preference Optimization [Rafailov et al., 2023] on paired contexts, where the higher-reward action is labeled as *preferred* and the lowest-reward action as *non-preferred*.
- **Head-based efficiency:** In PEARL and PEARL-Lite, the classification head consumes the last hidden state and outputs logits over the 8 actions, producing a one-token decision. This avoids autoregressive decoding and yields sub-20 ms inference on edge hardware.

These strategies allow us to study the trade-offs between reward alignment, training cost, and inference efficiency (see Section 4).

4 Experiments

4.1 Setup

Dataset. We construct a simulation dataset from WA measurements combined with synthetic application usage distributions. For each parameter tuple (`PerformanceMode`, `AccessCategory`), we measure average latency and power usage during controlled sessions. Each sample consists of (context, action, score), with scores computed as described in Section 3.2. Further details and parameter settings are provided in Table 4 and Appendix D.

Baselines. We evaluate our proposed framework for context-driven cross-layer optimization under the **PEARL** and **PEARL-Lite** variants introduced in Section 3.1. For comparison, we also include: (i) an *oracle*, which selects the configuration that maximizes the context-aware reward derived from ground-truth latency and energy — effectively representing the optimal balance between latency and energy consumption under the defined application constraints; (ii) *LoRA-only* (**LoRA**); (iii) a rule-based baseline (Rule), which selects WA parameters heuristically based only on application type (see Table 8 in Appendix E); and (iv) two fixed baselines, *Fixed (RT/IV)* (**fix-RT/IV**, (`realtime`, `interactiveVoice`)) and *Fixed (Bulk/BG)* (**fix-Bulk/BG**, (`bulk`, `background`)).

Metrics. We report three metrics aligned with the definitions in Section 3.2: (i) the **Objective Score**, corresponding to the overall reward $R(p)$ that combines normalized latency and energy; (ii) **Latency**, the latency score R_{lat} normalized by application-specific tolerance (higher is better); and (iii) **Energy**, the energy score R_{eng}, which reflects normalized energy consumption relative to device battery state (higher is better). Unless otherwise specified, results are averaged across scenarios covering different times of day and battery levels.

4.2 Performance Comparison

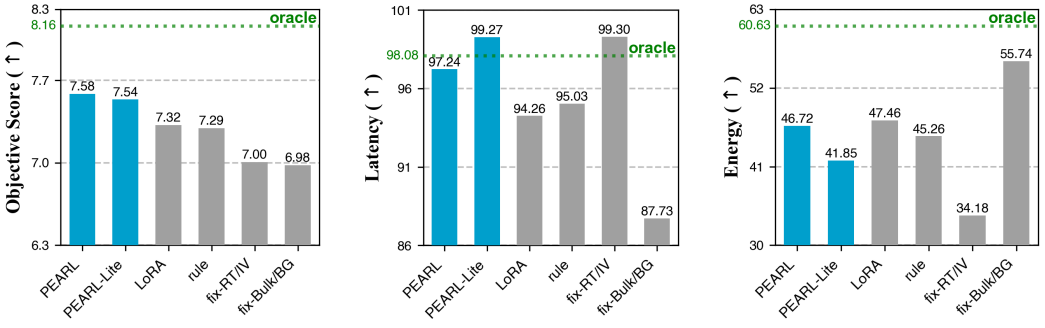


Figure 3: Objective score, latency, and energy comparison across PEARL variants and baselines.

Figure 3 shows that **PEARL** outperforms rule-based and fixed baselines on the joint objective score. The default **PEARL** variant achieves 7.58, with **PEARL-Lite** close behind at 7.54; both are about 7% below the oracle (8.16) but remain well above **LoRA-only** (7.32) and **Rule** (7.29). While **PEARL-Lite** emphasizes lower latency at the cost of higher energy, **PEARL** strikes a more balanced trade-off, leading to the best overall performance. These results demonstrate that head-based designs enable effective context-driven adaptation and substantially narrow the gap to the oracle compared to heuristic strategies.

4.3 Impact of Peer Context Information

Variant		Aggregated Scenario			Cooperative Scenario (Low-Battery Subscriber)		
		Objective Score ↑	Latency ↑	Energy ↑	Objective Score ↑	Latency ↑	Energy ↑
w/ peer info	(PEARL)	7.58	97.24	46.72	5.38	94.55	24.56
w/o peer info	(PEARL)	7.55	97.98	44.50	5.04	99.13	20.53
w/ peer info	(PEARL-Lite)	7.54	99.27	41.85	5.22	99.14	21.29
w/o peer info	(PEARL-Lite)	7.49	99.27	40.99	5.03	99.13	20.47

Table 1: Impact of including peer information in the context. In the aggregated scenario, models trained *w/ peer info* achieve consistently higher objective scores than those *w/o peer info*. In the cooperative scenario—where the subscriber is low on battery and the publisher adapts its configuration to assist—adding peer info reduces energy consumption by up to $\sim$16% while maintaining reasonable latency, highlighting where peer awareness provides the greatest benefit.

Table 1 shows that incorporating peer (subscriber) information consistently improves performance. In aggregated scenarios, PEARL and PEARL-Lite *w/ peer info* achieve slightly higher objective scores than their *w/o peer info* counterparts. In the cooperative case, **PEARL** *w/ peer info* reduces energy consumption by $\sim$16% while maintaining latency within 0.5% of *w/o peer info*, confirming that peer awareness provides tangible benefits in asymmetric conditions where coordination is critical. Additional qualitative snapshots illustrating this effect are provided in Appendix H.

4.4 Impact of Context-Aware Reward Design

Figure 4 shows that context-aware reward improves performance for both PEARL and PEARL-Lite. By normalizing latency against application-specific tolerances and incorporating device battery states into the energy term, the design generates richer supervision signals that better reflect heterogeneous requirements. Although absolute gains over the naive formulation are modest, they consistently yield higher objective scores, indicating more accurate adaptation. This demonstrates that careful reward design is crucial for leveraging context effectively in cross-layer optimization and for ensuring robustness across diverse usage scenarios.

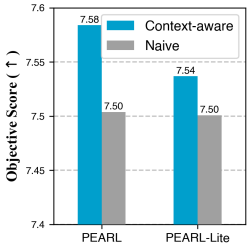
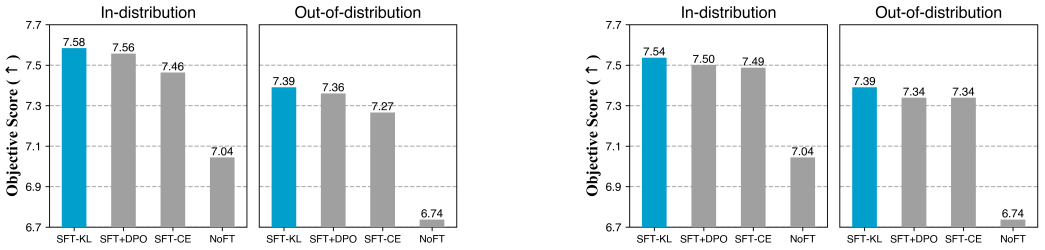


Figure 4: Objective scores with context-aware vs. naive reward.

4.5 Effect of Post-Training Strategy



(a) **PEARL** (b) **PEARL-Lite**

Figure 5: Objective scores for different training strategies. KL-based fine-tuning consistently outperforms CE and DPO, both in-distribution and out-of-distribution (OOD).

Figure 5 compares post-training strategies for **PEARL** and **PEARL-Lite**. Without fine-tuning (NoFT), objective scores are substantially lower in both in-distribution and OOD cases. Supervised fine-tuning with cross-entropy (SFT-CE) improves scores, while adding preference optimization (SFT+DPO) yields only marginal gains despite higher cost. In contrast, KL-based fine-tuning (SFT-KL) consistently achieves the best scores across both architectures. Its advantage stems from using soft labels derived from reward distributions, which preserve relative action preferences instead of collapsing them into hard labels. This provides richer supervision signals, enabling more nuanced trade-offs and better generalization across diverse scenarios.

4.6 Efficiency of Post-Training Modules

Baseline	Objective Score ↑	Inference ↓			Training ↓	
		Time (ms)	Tokens	RAM (GB)	RAM (GB)	Time (min/epoch)
Base	7.04	760.7 (×1.0)	48.9	6.58	—	—
LoRA-only	7.32	243.1 (×3.1)	7.62	9.39	29.17	30
PEARL-Lite	7.54	**14.5 (×52)**	1.00	6.80	11.03	12
PEARL	**7.58**	40.3 (×18.9)	1.00	9.44	46.83	38

Table 2: Performance and efficiency of post-training modules. Parentheses denote speedup vs. Base (NoFT) . "Tokens" = average generated tokens per decision; head-based models always produce 1.00 (classification head). **PEARL-Lite** is most efficient, while **PEARL** yields the highest objective score.

Table 2 highlights the trade-off between performance and efficiency across different post-training modules. **PEARL** achieves the highest objective score (7.58) but incurs moderate training cost, while **PEARL-Lite** attains nearly comparable performance (7.54) with substantially lower inference latency (14.5 ms, 52× faster than the Base) and reduced training memory. In contrast, **LoRA-only** yields lower objective scores (7.32) and slower inference, underscoring that head-based designs provide the most favorable balance for on-device deployment. This efficiency comes from the classification head: it consumes the last token's hidden state to produce logits over the 8-action space (a single token), whereas LoRA-only relies on text generation, producing ∼8 tokens on average.

5 Discussion

Q1 — Interpretation: ***How do peer context and a context-aware reward improve PEARL's on-device D2D decisions?*.** Incorporating subscriber-side context reduces partial observability and enables coordinated adaptation across devices. This leads to consistently higher objective scores, and in cooperative cases (*e.g.*, low-battery subscriber) reduces energy by ∼16% with negligible latency change (≤0.5%) (Table 1), showing that peer awareness makes PEARL more responsive to asymmetric conditions. The context-aware reward amplifies this effect by aligning training with heterogeneous constraints: latency is normalized by application-specific tolerances, and energy is scaled according to device battery states (Sec. 3.2, Fig. 4, Appendix C). Together, these choices guide PEARL toward energy-efficient yet latency-safe actions, improving robustness in asymmetric scenarios.

Q2 — Practical Implications: ***How does PEARL's head-based design enable efficient deployment on real edge devices?*.** PEARL's head-based design performs inference by emitting a *single* classification token from the last hidden state, avoiding autoregressive decoding and its latency/memory overhead. This allows **PEARL-Lite** to run in under 20 ms (∼52× faster than the base) with low memory usage, while full **PEARL** achieves the highest objective score with roughly 3× inference latency and 3–4× training memory relative to PEARL-Lite — a trade-off that yields measurable gains in accuracy and robustness. By contrast, LoRA-only is slower and less effective due to multi-token generation. In deployment, **PEARL-Lite** is well-suited when latency or GPU memory (VRAM) budgets are tight, **PEARL** is preferable when slight overhead is acceptable for maximum accuracy, and merging adapters into the base can approximate PEARL's performance at near PEARL-Lite cost (Table 2). Overall, these properties give PEARL the latency–energy–memory profile required for always-on, on-device cross-layer control without server support.

Q3 — Alternatives: ***Why not replace PEARL with a compact feed-forward network (FFN/MLP), a reinforcement-learning (RL) baseline, or a lightweight Transformer encoder?*.** While small feed-forward networks or RL baselines can be efficient for narrow tasks, they struggle with mixed-format context (numeric + categorical) and typically require additional feature engineering. Lightweight Transformer encoders such as `MiniLM` (22.7M parameters) [Wang et al., 2020] can process heterogeneous inputs more flexibly and achieve very low inference latency (∼5–6 ms). However, as Table 3 shows, they yield lower objective and OOD scores than PEARL. By contrast, PEARL and PEARL-Lite leverage pretrained language modeling to naturally parse diverse context, providing stronger generalization while still running in the sub-20 ms range. Thus, although compact models may be attractive under extreme latency budgets, PEARL offers the best balance of accuracy, robustness, and deployability for always-on D2D optimization.

Baseline	Objective Score ↑	OOD Score ↑	Inference Time (ms) ↓
PEARL	**7.58**	**7.39**	40.3
PEARL-Lite	7.54	**7.39**	14.5
MiniLM (Head+LoRA)	7.51	7.34	6.0
MiniLM (Head)	7.49	7.34	**5.1**

Table 3: Comparison of PEARL with a lightweight Transformer encoder (`all-MiniLM-L6-v2`, 22.7M).

Q4 — Limitations & Future Directions. We provide a proof-of-concept demo, with code, dataset, and a full demonstration video available at `https://github.com/abman23/pearl` to ensure reproducibility. Nevertheless, rigorous validation on hardware testbeds with real-time counters (*e.g.*, per-packet latency and energy) is required for robust evaluation. Our study is currently limited to a narrow task space (2 `PerformanceMode` options $\times$ 4 `AccessCategory` options) and a single protocol (WA). Extending PEARL to richer parameter spaces, additional protocols (*e.g.,* 5G), and heterogeneous devices remains an open challenge. Moreover, while head-based inference provides fast and accurate decisions, scaling to multiple tasks with separate heads may become burdensome. Future work should explore multi-task optimization strategies that preserve efficiency while maintaining accuracy.

References

Apple Inc. Apple intelligence foundation language models, 2024. URL https://arxiv.org/abs/2407.21075.

M. J. J. Bucher and M. Martini. Fine-tuned LLMs (still) significantly outperform zero-shot generative AI models in text classification. *arXiv preprint arXiv:2406.08660*, 2024. URL https://arxiv.org/abs/2406.08660.

D. Camps-Mur, E. Garcia-Villegas, E. Lopez-Aguilera, P. Loureiro, and et al. Enabling always-on service discovery: Wi-Fi neighbor awareness networking. *IEEE Wireless Commun.*, 22(2):118–125, 2015. doi: 10.1109/MWC.2015.7096294.

Z. Han, C. Gao, J. Liu, J. Zhang, and et al. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024. URL https://arxiv.org/abs/2403.14608.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531 [cs.CL]*, 2015. URL https://arxiv.org/abs/1503.02531.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, and et al. LoRA: Low-rank adaptation of large language models. In *Int. Conf. on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2106.09685.

J.-H. Lee, Y. Lu, and K. Doppler. On-device LLM for context-aware Wi-Fi roaming. *arXiv preprint arXiv:2505.04174*, 2025. URL https://arxiv.org/abs/2505.04174.

G. Qu, Q. Chen, W. Wei, Z. Lin, and et al. Mobile edge intelligence for large language models: A contemporary survey. *IEEE Commun. Surveys & Tutorials*, 2025.

R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, and et al. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:53728–53741, 2023.

J. Shao, J. Tong, Q. Wu, W. Guo, and et al. WirelessLLM: Empowering large language models towards wireless intelligence. *arXiv preprint arXiv:2405.17053*, 2024. URL https://arxiv.org/abs/2405.17053.

W. Wang, F. Wei, L. Dong, H. Bao, and M. Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020. URL https://arxiv.org/abs/2002.10957.

D. Wu, X. Wang, Y. Qiao, Z. Wang, and et al. NetLLM: Adapting large language models for networking. In *Proc. ACM SIGCOMM*, pages 661–678, 2024.

A Real-World Demonstration

To validate the feasibility of **PEARL** beyond simulation, we implemented a real-world prototype as an `iPadOS` app. The app establishes D2D communication between two devices using Apple's WA framework and integrates Apple's on-device Foundation Models (AFM) to run the LLM agent directly on the publisher [Apple Inc., 2024]. The agent monitors local and peer-side context and dynamically updates WA parameters (`PerformanceMode`, `AccessCategory`) to reconfigure the ongoing connection. We deploy the app on two iPad Pro devices (Apple M4 chip, 8 GB RAM, `iPadOS` 26.0). A full demonstration video is available at `https://github.com/abman23/pearl`.

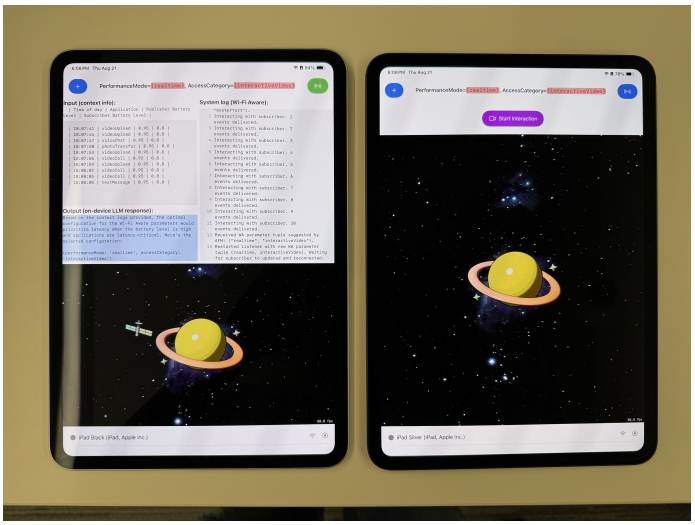


Figure 6: Prototype demonstration of PEARL on two iPad Pro devices. The publisher (left) hosts the on-device LLM agent, while the subscriber (right) receives updated WA parameters.

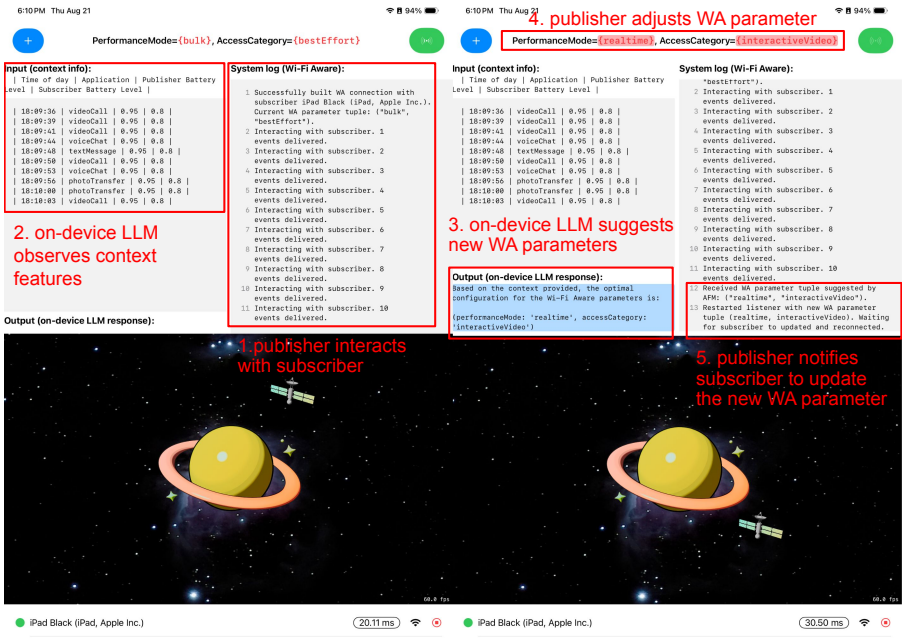


Figure 7: Step-by-step illustration of WA parameter tuning. The publisher observes context features and proposes a new configuration (`realtime`, `interactiveVideo`), then reconfigures the WA link and notifies the subscriber to update.

B Setup

Table 4 summarizes the hardware, software, dataset, and training configurations. For task design, we synthesize eight application types with time-of-day usage distributions that mimic typical mobile usage patterns (Fig. 8a). These distributions are used to sample application types as part of the context features in training and in-distribution evaluation. To test generalization, we construct an alternative distribution (Fig. 8b) to generate out-of-distribution (OOD) evaluation sets.

HW / SW	
Device (Publisher / Subscriber)	iPad Pro (M4, 8 GB RAM)
OS	`iPadOS 26.0`
Frameworks	`Wi-Fi Aware`, `Foundation Models`
LLM Backbone	`Llama-3.2-3B`, `AFM-on-device` [Apple Inc., 2024]
Dataset (WA Log)	
Total Samples	32,000
Train / Test Split	80% / 20%
Sampling Interval	5 s
Task Configuration	
Action Space	2 modes × 4 categories = 8 actions
Scenarios	16 combinations of time-of-day (morning, afternoon, evening, night) and device battery levels (both high, medium, low, publisher high / subscriber low)
Application Types	`textMessage`, `voiceChat`, `videoCall`, `sensorSync`, `photoTransfer`, `videoUpload`, `firmwareUpdate`, `mapSync`
Contextual Inputs	Time, App Types, Publisher Battery Level, Subscriber Battery Level
Reward Weights	$w_L = 0.1$, $w_P = 1.0$
History Window	10 past time steps
Training Configuration	
Learning Rate	1×10^{-5}
Batch Size / Grad. Accum.	4 / 16 (effective batch = 64)
Epochs	5
Weight Decay	0.01
Loss Functions	CE, KL, DPO
LoRA Rank / Alpha	128 / 128
DPO Beta	0.1
Optimizer	`AdamW`
GPU	1 × NVIDIA L40S

Table 4: Experiment setup including hardware, dataset, task configuration, and training details.

```
morning:
    textMessage: 0.25,
    voiceChat: 0.20,
    videoCall: 0.25,
    mapSync: 0.20,
    photoTransfer: 0.10
afternoon:
    textMessage: 0.20,
    voiceChat: 0.20,
    videoCall: 0.25,
    sensorSync: 0.20,
    photoTransfer: 0.15
evening:
    textMessage: 0.10,
    voiceChat: 0.20,
    videoUpload: 0.30,
    photoTransfer: 0.20,
    videoCall: 0.20
night:
    firmwareUpdate: 0.40,
    sensorSync: 0.30,
    textMessage: 0.10,
    photoTransfer: 0.10,
    mapSync: 0.10
```

(a) Training / in-distribution

```
morning:
    videoCall: 0.30,
    textMessage: 0.25,
    voiceChat: 0.20,
    photoTransfer: 0.15,
    videoUpload: 0.10
afternoon:
    videoCall: 0.25,
    textMessage: 0.25,
    voiceChat: 0.20,
    photoTransfer: 0.20,
    videoUpload: 0.10
evening:
    videoUpload: 0.35,
    videoCall: 0.25,
    photoTransfer: 0.20,
    voiceChat: 0.15,
    textMessage: 0.05
night:
    videoUpload: 0.30,
    videoCall: 0.25,
    textMessage: 0.20,
    voiceChat: 0.15,
    photoTransfer: 0.10
```

(b) Out-of-distribution evaluation

Figure 8: Synthetic application usage distributions used to generate context logs.

C Context-Aware Evaluation Metrics

We define two context-dependent metrics, **Latency** and **Energy**, to evaluate link quality under D2D dynamics. Unlike raw values, these metrics incorporate application- and device-specific constraints, providing not only quantitative measurement but also qualitative insight into how well system behavior aligns with user needs.

Latency is computed as a function of WA parameters and application type, expressed as a percentage score indicating how well the observed link latency satisfies the *latency tolerance* of the active application (Table 5). **Energy** is computed as a function of WA parameters and device battery level, reflecting how well the energy consumption of a configuration aligns with the current battery status.

Application Type	Latency Tolerance (ms)
`textMessage`	200
`voiceChat`	50
`videoCall`	100
`sensorSync`	1,000
`photoTransfer`	2,000
`videoUpload`	5,000
`firmwareUpdate`	10,000
`mapSync`	500

Table 5: Latency tolerance thresholds for each application type.

To highlight the benefit of these metrics, we contrast them with raw measurements. Table 6 shows that while raw latency (ms) varies significantly across time periods, the **Latency** score remains consistent with application tolerance. For example, although average link latency at night (15.61 ms) is much higher than in the morning (4.97 ms), most nighttime applications are delay-tolerant, producing similarly high scores. Likewise, Table 7 compares **Energy** with raw energy consumption across high, medium, and low battery states. Here, the metric magnifies differences under low-battery conditions, assigning greater penalty and dynamically shifting the optimization objective toward energy efficiency.

Scenario (time, battery)	Latency Score ↑	Link Latency (ms) ↓
(morning, both medium)	98.41	4.97
(afternoon, both medium)	98.43	6.60
(evening, both medium)	98.94	9.56
(night, both medium)	98.95	15.61

Table 6: Comparison of context-aware **Latency** score and raw link latency across scenarios.

Scenario (time, battery)	Energy Score ↑	Energy Consumption (%/h) ↓
(afternoon, both high)	102.99	3.15
(afternoon, both medium)	59.46	3.24
(afternoon, both low)	24.26	2.90

Table 7: Comparison of context-aware **Energy** score and raw energy consumption across scenarios.

D Data Collection

The simulation dataset is constructed from real WA logs, combined with synthetic application usages defined in Fig. 8. To collect logs, we developed an `iPadOS` app that establishes a WA connection between two devices, simulates their interactions, and records context information. The app was deployed on two iPad Pro devices (Table 4), where we executed 16 sessions of data collection, each consisting of 2,000 consecutive logs. Each session corresponds to one specific scenario defined by time of day and device battery levels. Battery levels were sampled uniformly across high, medium, and low states to ensure balanced representation of energy conditions.

During each session, we iterated over all WA parameter tuples in the action space so that every configuration (`PerformanceMode`, `AccessCategory`) was exercised. For each tuple, the app recorded both link latency values and battery usage, which are later used to compute the objective score as described in Section 3.2 and Section C.

Logged features. The following features were logged per record:

- Device context: publisher and subscriber device IDs, battery levels, time of day, application type.
- WA parameters: current `PerformanceMode` and `AccessCategory`.
- Performance metrics: measured latency and battery usage.

Additional fields such as charging status, throughput capacity, signal strength, and throughput-related counters were also recorded (Figs. 9, 10), but were not used for training or inference. These remain available for future extensions or cross-validation.

Figure 9: Data collection pipeline. WA logs are recorded on the publisher and uploaded to a computer for pre-processing.

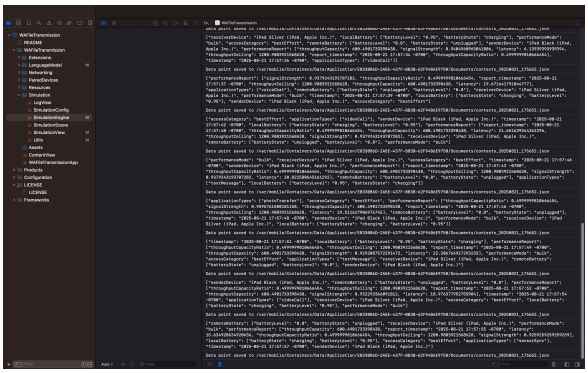


Figure 10: Example WA logs received by the computer, including device context, WA parameters, and performance metrics.

E Baseline Details (Rule)

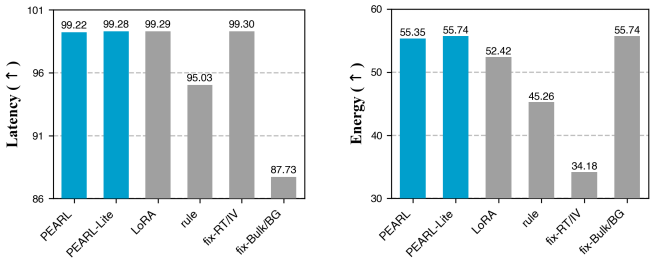
The **Rule** baseline is a simple heuristic policy that depends only on application type. For each application, we assign a preferred WA parameter tuple based on common-sense intuition about its latency or throughput requirements (Table 8). At inference time, the baseline examines the past n applications (with $n = 10$ in our setup, consistent across all baselines) and selects the parameter tuple that appears most frequently among their preferred mappings.

This design makes **Rule** adaptive only to recent application history, without considering device battery states or other contextual factors. It thus serves as a lightweight non-learning baseline for comparison against context-aware methods such as PEARL and PEARL-Lite. Future extensions could incorporate a simple "battery-safe" heuristic—e.g., switching to energy-preserving modes below 20% battery—to better reflect real-world smartphone behavior.

Application Type	Preferred Parameter Tuple
`textMessage`	(`realtime`, `bestEffort`)
`voiceChat`	(`realtime`, `interactiveVoice`)
`videoCall`	(`realtime`, `interactiveVideo`)
`sensorSync`	(`bulk`, `background`)
`photoTransfer`	(`bulk`, `bestEffort`)
`videoUpload`	(`bulk`, `bestEffort`)
`firmwareUpdate`	(`bulk`, `background`)
`mapSync`	(`realtime`, `bestEffort`)

Table 8: Preferred WA parameter tuple assigned heuristically for each application type.

F Single-Objective Optimization Results



(a) Latency-only (b) Energy-only

Figure 11: Performance comparison of PEARL variants and baselines for single-objective optimization.

Although PEARL is primarily designed for the joint optimization of latency and energy, we also evaluate performance under single-objective settings. Figure 11 compares PEARL variants against baselines when optimizing for **Latency** only or **Energy** only.

In the latency-only setting (Fig. 11a), **PEARL**, **PEARL-Lite**, and **LoRA-only** achieve scores within 1% of **Fixed(RT/IV)**, which always selects (`realtime`, `interactiveVoice`)—a configuration strongly favoring low latency. In the energy-only setting (Fig. 11b), **PEARL** and **PEARL-Lite** achieve performance close to **Fixed(Bulk/BG)**, the parameter choice that minimizes energy consumption, while outperforming other adaptive baselines.

Interestingly, **PEARL-Lite** performs on par with, and sometimes slightly better than, full **PEARL** in single-objective settings. This suggests that lightweight head-only models may be particularly well-suited for simpler optimization tasks, offering both efficiency and strong performance.

G Head Module Design Details

The head module in **PEARL-Lite** (and **PEARL**) is responsible for mapping the last-token embedding of the frozen LLM encoder to logits over the WA action space. Because this component is lightweight compared to the backbone LLM, its depth can be adjusted without incurring meaningful additional cost. We therefore explore different head architectures to examine whether deeper designs improve representational power.

Variant	Performance ↑			Inference ↓		Training ↓	
	Objective Score	Latency	Energy	Time (ms)	RAM (GB)	Time (min/epoch)	RAM (GB)
3-layer Head	7.54	99.27	41.85	14.5	6.80	12	11.03
2-layer Head	7.50	99.27	41.21	14.6	6.80	12	10.54
1-layer Head	7.51	99.27	41.35	14.7	6.80	12	11.30

Table 9: Performance and efficiency of different head module depths in **PEARL-Lite**.

As shown in Table 9, the 3-layer head—the default configuration—achieves the highest objective score, while maintaining identical inference and training efficiency compared to shallower variants. This result suggests that slightly deeper heads can better utilize the LLM's feature representations, while the overall system cost remains dominated by the backbone. Hence, deeper heads are a simple but effective design choice for improving performance without compromising efficiency.

H Full Scenario Snapshots

To complement the quantitative analysis in Section 4.3, we present two qualitative snapshots of context logs and the corresponding predictions made by **PEARL**, **PEARL w/o peer info**, and **Rule**. These examples illustrate how peer-side context guides parameter selection toward more energy-aware configurations when the subscriber is low on battery.

```
Sample 181, context feature:
| Time of day | Application | Publisher Battery Level | Subscriber Battery Level |
| 16:53:13 | ['sensorSync'] | 0.75 | 0.1 |
| 16:54:03 | ['videoCall'] | 0.75 | 0.1 |
| 16:54:53 | ['sensorSync'] | 0.75 | 0.1 |
| 16:55:43 | ['photoTransfer'] | 0.75 | 0.1 |
| 16:56:33 | ['videoCall'] | 0.75 | 0.1 |
| 16:57:23 | ['textMessage'] | 0.75 | 0.1 |
| 16:58:13 | ['photoTransfer'] | 0.75 | 0.1 |
| 16:59:03 | ['textMessage'] | 0.75 | 0.1 |
| 16:59:53 | ['photoTransfer'] | 0.75 | 0.1 |
| 17:00:43 | ['textMessage'] | 0.75 | 0.1 |
PERAL prediction: ('bulk', 'background'), objective score: 6.346243878475599
PEARL w/o peer info prediction: ('realtime', 'bestEffort'), objective score: 4.211099506296795
Rule prediction: ('bulk', 'bestEffort'), objective score: 3.9404831260206663
```

(a) Afternoon scenario.

```
Sample 135, context feature:
| Time of day | Application | Publisher Battery Level | Subscriber Battery Level |
| 21:50:38 | ['videoUpload'] | 0.75 | 0.1 |
| 21:51:28 | ['photoTransfer'] | 0.75 | 0.1 |
| 21:52:18 | ['photoTransfer'] | 0.75 | 0.1 |
| 21:53:08 | ['videoCall'] | 0.75 | 0.1 |
| 21:53:58 | ['videoUpload'] | 0.75 | 0.1 |
| 21:54:48 | ['voiceChat'] | 0.75 | 0.1 |
| 21:55:38 | ['videoCall'] | 0.75 | 0.1 |
| 21:56:28 | ['videoUpload'] | 0.75 | 0.1 |
| 21:57:18 | ['videoCall'] | 0.75 | 0.1 |
| 21:58:08 | ['videoCall'] | 0.75 | 0.1 |
PERAL prediction: ('bulk', 'background'), objective score: 8.151432251908401
PEARL w/o peer info prediction: ('realtime', 'bestEffort'), objective score: 5.070268562892394
Rule prediction: ('bulk', 'bestEffort'), objective score: 7.964587164826415
```

(b) Evening scenario.

Figure 12: Context logs and decisions of **PEARL**, **PEARL w/o peer info**, and **Rule** under the subscriber low-battery case.

In Fig. 12a, when the subscriber battery is only 10%, **PEARL** leverages this information to select `(bulk, background)`, prioritizing energy saving. Without access to peer context, **PEARL w/o peer info** observes only the publisher's high battery level and instead selects `(realtime, bestEffort)`. **Rule**, based solely on application type, chooses `(bulk, bestEffort)`, yielding the lowest objective score.

Fig. 12b shows a complementary case from the evening period. Here, despite a mix of latency-sensitive and high-throughput applications such as `videoCall` and `videoUpload`, **PEARL** again adapts to the low-battery subscriber and opts for `(bulk, background)`. Both **PEARL w/o peer info** and **Rule** fail to adjust accordingly, resulting in lower scores.

Taken together, these snapshots demonstrate that **PEARL** consistently leverages peer-side context across different scenarios, leading to more energy-aware decisions while maintaining strong performance.