2026 Volume: 02 Issue: 07 www.ijrpa.com ISSN 2456-9995 Research Article

International Journal Research Publication Analysis

Page: 01-13

FEDERATED LEARNING FOR PRIVACY-PRESERVING WORD PREDICTION IN REAL-TIME MESSAGING APPLICATIONS: A FEDAVG-BASED FRAMEWORK WITH DIFFERENTIAL PRIVACY

***Dr. Somendra Kumar, Kriti Bansal, Dr. Nitin Saraswat**

Assistant Professor, Department of Information Technology, Jagan Institute of Management Studies (JIMS), Rohini, New Delhi, India.

Article Received: 26 May 2026
Article Revised: 15 June 2026
Published on: 04 July 2026

***Corresponding Author: Dr. Somendra Kumar**
Assistant Professor, Department of Information Technology, Jagan Institute of Management Studies (JIMS), Rohini, New Delhi, India.

DOI: https://doi-doi.org/101555/ijrpa.7282

ABSTRACT

Word prediction has become a routine feature of real-time messaging platforms, yet the models behind it are usually trained the same way: text typed by users is logged, pooled on a server, and used to fit a language model centrally. That approach works, but it sits uneasily next to what people expect from a private conversation. This paper works through an alternative — a federated learning (FL) pipeline in which the word-prediction model is trained directly on each user's device, and only a bounded, noised summary of what changed locally is ever sent to a server. We call the noised variant DP-FedAvg: standard Federated Averaging with per-client update clipping and Gaussian noise layered on top. Since production chat logs could not ethically be collected for this study, we instead built a controlled testbed from a public, dialogue-structured text corpus, splitting it by speaker into 24 simulated clients whose vocabulary and style differ enough to behave like genuinely non-IID users. A lightweight neural language model was then trained under four regimes — centralized pooling, plain FedAvg, DP-FedAvg at three noise levels, and fully isolated local training — and evaluated on next-word accuracy and perplexity. Centralized training tops out at 7.23% top-1 / 20.51% top-5 accuracy after 8 epochs; FedAvg reaches 4.50% / 14.84% after 15 rounds without a single message ever leaving a device; and adding differential-privacy noise at a moderate level ($\sigma = 0.05$) costs only about 0.2 points more. A less expected result is that FedAvg barely outperforms training each client in isolation once the data is this heterogeneous — a finding we treat as a pointer toward personalization rather than a flaw in the privacy mechanism. We close with a discussion of how this design would slot into an

existing MERN/Socket.IO chat stack via on-device TensorFlow.js training and a Node.js aggregation service, along with the gaps that remain before it could move from a research prototype to something deployable.

KEYWORDS: Federated Learning, Privacy-Preserving AI, Word Prediction, Real-Time Messaging, Differential Privacy, FedAvg, Natural Language Processing, On-Device Learning, Non-IID Data, Edge AI, MERN Stack.

I. INTRODUCTION

Most people now expect a messaging app to finish their sentences for them, or at least offer a shortcut. In earlier work, we built exactly this feature into a MERN-stack chat application (MongoDB, Express.js, React.js, Node.js) with Socket.IO handling delivery, and trained the underlying TensorFlow word-suggestion model on a mix of logged chat data and open corpora [8]. It performed well — that is the pattern most production systems follow, after all. But it also meant that raw or lightly anonymized user text had to reach a central server before the model could learn anything from it, which is not an assumption most people make when they open a chat app to talk to a friend or a colleague.

Federated learning offers a way out of that assumption. Instead of collecting text, an FL system trains a shared model across many devices and only ever exchanges the resulting parameter updates with a server [2]. This is not a hypothetical fix — Google's Gboard keyboard has run exactly this kind of pipeline in production for next-word prediction for years, training a recurrent model with the Federated Averaging (FedAvg) algorithm directly on phones [1], and the same technique was later extended to query-suggestion ranking [3]. So the question this paper asks is not whether federated word prediction is feasible — it clearly is — but whether it can be built and evaluated for a messaging context specifically, and what it costs in accuracy to do so responsibly.

That responsibility matters because FedAvg by itself is not a privacy guarantee. Suliman and Leith [6] showed that the model updates exchanged during federated training of text models can be reversed with targeted reconstruction attacks, recovering the words a user actually typed — and that ad hoc defenses like mini-batching or unstructured noise do not stop this. A more principled fix, demonstrated by McMahan et al. [4], is to clip each client's update and add carefully calibrated Gaussian noise before it ever leaves the device, giving the aggregation something closer to a formal differential-privacy guarantee. That is the mechanism this paper builds on.

With that motivation in place, this paper makes four contributions:

- A federated word-prediction architecture for real-time messaging that extends the MERN/Socket.IO chat design of [8] with on-device training, per-client gradient clipping, and Gaussian noise injection (DP-FedAvg), so raw message text never leaves a user's device.
- A controlled, reproducible experimental testbed built from a publicly available dialogue-structured text corpus, partitioned by speaker into 24 non-IID simulated clients, with a full FedAvg / DP-FedAvg training and evaluation pipeline implemented from first principles rather than borrowed off the shelf.
- A head-to-head comparison of centralized (data-pooled) training, plain FedAvg, DP-FedAvg at three noise levels, and fully isolated local-only training, reported in top-1/top-5 next-word accuracy and perplexity.
- An analysis of the resulting privacy-utility tradeoff, the communication cost of the architecture, and what it would actually take to fold this into a production MERN chat stack.

II. Related Work

A. Real-Time Messaging and Centralized Word Suggestion

Raj and Saraswat [8] built a MERN-stack chat application with real-time word suggestion powered by a TensorFlow model, trained on a custom chat dataset alongside the Cornell Movie Dialogs and Reddit Comments corpora. The reported numbers were solid — 85% precision, 78% recall, suggestions delivered in under 300ms — but the training pipeline behind them pooled every logged message on a central server. That is the gap this paper tries to close: the same messaging context and the same underlying problem, but a training approach that never requires the text to leave the user's device in the first place.

B. Federated Learning Fundamentals

McMahan et al. [2] introduced FedAvg to answer a fairly specific question: how do you train a shared model when the data cannot be centralized? Their answer was to let each client run several local epochs of SGD from the current global weights, then average the results back on the server, weighted by how much data each client contributed. What matters for a messaging use case is that this held up reasonably well even when client data was unbalanced and non-identically distributed — close to what you would expect across users with very different vocabularies and typing habits. Kairouz et al. [5] later surveyed how the field developed from

there, cataloguing open problems around communication cost, device heterogeneity, and privacy that shape several of the design choices made in this paper.

C. Federated Learning for Text and Word Prediction

The clearest evidence that this approach works at scale comes from Hard et al. [1], who trained an RNN language model for Gboard's next-word prediction using FedAvg on real client devices, and found that training on higher-quality, naturally occurring on-device data could match or beat a server-trained model on recall — without ever exporting what users typed. Yang et al. [3] pushed the same idea into query suggestion, and their paper is useful here for a more practical reason: it describes the system-level rules used to decide when a device may participate in a training round (idle, charging, on an unmetered connection), details that shape the deployment discussion in Section VI.

D. Privacy Risks and Defenses in Federated NLP

None of this makes FL automatically private. Suliman and Leith [6] demonstrated that the very updates FedAvg is built around can leak the text that produced them, through local reconstruction attacks against the Gboard pipeline, and found that the obvious countermeasures — mini-batching, informal local noise — do not hold up. McMahan et al. [4] took a more rigorous path: clip each client's contribution, add Gaussian noise calibrated against a formal privacy budget, and track the cumulative privacy loss with a moments accountant. Their finding — that recurrent language models trained this way stay close in quality to non-private ones, given enough clients — is the result this paper's DP-FedAvg design is modeled on, though as Section VII makes clear, the full accounting machinery is not implemented here.

E. Federated Learning in Other Privacy-Sensitive Domains

The same tension — model quality versus data centralization — shows up well outside messaging. Xu et al. [7] discuss it in the context of healthcare informatics, where patient records make centralized training even harder to justify. Seeing the same motivation recur across such different domains is part of why FL is treated here less as a messaging-specific trick and more as a general pattern worth applying wherever the underlying data is personal.

III. Threat Model and Design Goals

We assume an honest-but-curious server: it follows the FedAvg protocol correctly, but might try to extract information about a client's text from the updates it receives — exactly the setting Suliman and Leith [6] exploit. Four design goals follow from that assumption. Raw message text should never leave the device, in any form. Each client's contribution to the

aggregate should be bounded, via norm clipping, so no single user's update can dominate — or be too easily isolated from — the aggregation. Calibrated noise should be layered on top of that clipped update before transmission, following the pattern set out in [4]. And after all of that, the resulting model still has to be good enough that suggesting words in real time feels useful rather than gimmicky.

IV. Proposed System Architecture

Figure 1 sketches the architecture. Each client keeps its chat history — or some rolling window of recent messages — entirely on-device, and trains a local copy of the shared word-prediction model against it. Nothing about that text gets uploaded. Instead, the client works out what changed: the difference between its freshly trained weights and the global weights it started the round with. That difference gets clipped to a fixed norm, has noise added to it, and only then gets sent to the server. The server folds together whatever updates arrive in a given round using a data-weighted FedAvg rule, and sends the resulting model back out for the next round.

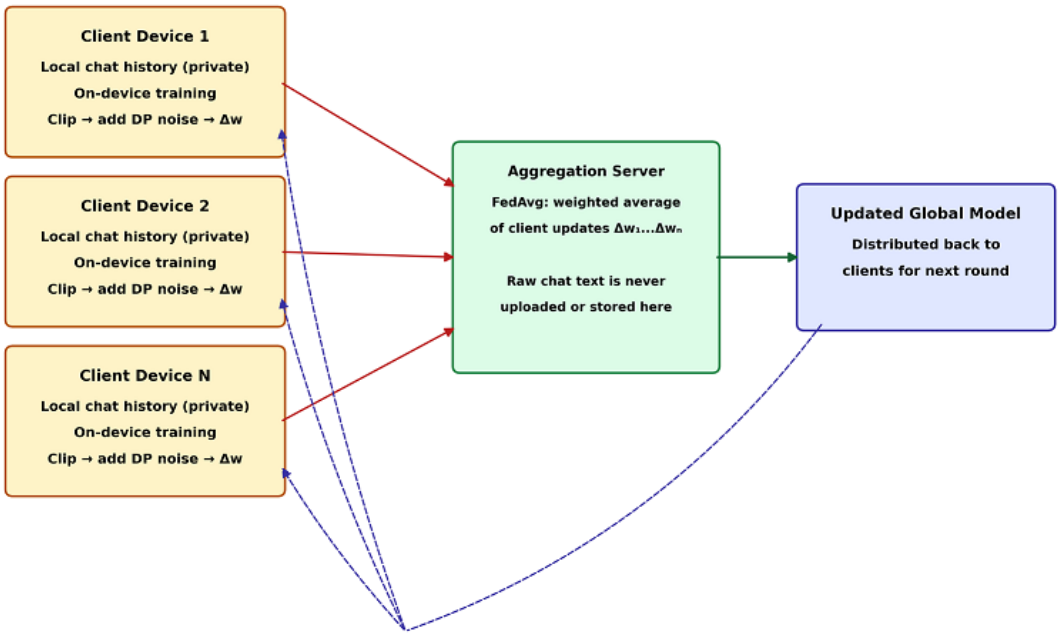

Figure 1: Federated word-prediction architecture for a real-time messaging application.

Figure 2 walks through what one of those rounds actually looks like: the server broadcasts current weights, each participating client trains locally for a few epochs, the client clips and noises its update, the server aggregates everything it received, and the cycle repeats — either for a fixed number of rounds or until validation loss stops improving.

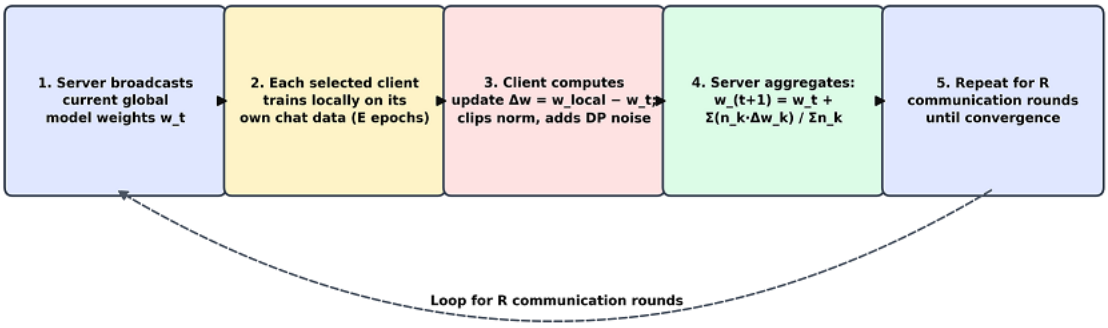

Figure 2: One communication round of DP-FedAvg training.

Plugging this into the chat application from [8] would mean retiring the centralized TensorFlow training loop on the Express backend and replacing it with two things: a lightweight TensorFlow.js model running client-side (in the React web app or a React Native equivalent) for both inference and local training, and a Node.js/Express endpoint that receives clipped, noised weight updates over HTTPS or a WebSocket and runs the aggregation step. MongoDB's role shrinks to storing successive versions of the global model — never message text. Socket.IO keeps doing exactly what it already does for message delivery; only the training path for the suggestion model changes.

V. METHODOLOGY

A. Dataset and Non-IID Client Partitioning

Collecting real chat logs for this study was not an ethical option, so a stand-in was needed. We settled on a publicly available, dialogue-structured English corpus — the "Tiny Shakespeare" text often used to prototype language models [9] — precisely because it is organized as labeled speaker turns rather than flat prose. That structure allows it to be partitioned in a way that is structurally similar to per-user chat data: each character's lines become one simulated client's private, on-device text, with no overlap or sharing between clients. We kept the 24 speakers with at least 60 turns each (2,826 turns total), which gives a partition that is genuinely non-IID — these characters do not just speak about different things, they use noticeably different vocabulary and sentence patterns, roughly the kind of heterogeneity expected across real users. It is worth being upfront that this is a literary corpus standing in for conversational text, not the real thing; we return to that limitation in Section VII along with a plan to revalidate on real, opt-in, anonymized chat data — the same kind of corpora (Reddit Comments, Cornell Movie Dialogs) used in [8].

Table 1: Dataset and model configuration for the federated word-prediction testbed.

Property	Value
Simulated clients (speakers)	24
Total dialogue turns used	2,826
Vocabulary size	2,000 (+ <unk>, <pad>)
Context window	3 preceding words
Pooled training examples	63,338
Pooled test examples	11,163
Model parameters	200,208 (≈ 0.76 MB, fp32)

B. Preprocessing

Text from each client is lowercased, stripped of punctuation apart from apostrophes, and split into words. From the pooled vocabulary across all 24 clients, the 1,998 most frequent words are kept, plus reserved <unk> and <pad> tokens, giving a 2,000-word vocabulary. Each client's word stream is then converted into training examples using a sliding 3-word context window to predict the next word, padding the start of the stream as needed. Examples where the target word falls outside the vocabulary are dropped, since predicting <unk> is not a meaningful task. Each client's examples are split 85/15 into local train and test sets, yielding 63,338 training examples and 11,163 test examples pooled across all 24 clients.

C. Model Architecture

A small feed-forward language model was used rather than a recurrent one, and that choice was deliberate: on-device federated training has to fit within a mobile device's compute, memory, and battery budget, a constraint the FL literature keeps coming back to [5]. The model embeds each of the 3 context words into a 32-dimensional vector, concatenates them, passes the result through a 64-unit ReLU hidden layer, and produces a softmax over the 2,000-word vocabulary — 200,208 parameters in total, about 0.76 MB in 32-bit floats. That is small enough to sync over a normal mobile connection, which lines up with how Google restricts Gboard's federated training rounds to devices that are idle, charging, and on unmetered Wi-Fi [3]. A production system would probably use something closer to the coupled-input-forget-gate LSTM from [1] for better sequence modeling; the architecture here was kept this simple on purpose, so that what is being measured is the effect of FedAvg and DP-FedAvg themselves, not differences in model capacity.

D. Training Regimes Compared

- Centralized — every client's examples pooled on one server, trained with ordinary mini-batch SGD for 8 epochs. This is the upper-bound baseline, and also the one that violates the privacy assumption the rest of the paper is built around.
- FedAvg — the standard algorithm [2], run across the 24 clients for 15 communication rounds with 1 local epoch per client per round, aggregated by a data-weighted average. No raw text, and no un-noised updates, are ever centralized.
- DP-FedAvg — the same FedAvg setup, but each client's update is clipped to an L2 norm of 1.0 and perturbed with Gaussian noise at one of three levels (σ = 0.01, 0.05, 0.1) before it is sent, following the clip-and-noise pattern from [4].
- Local-only — each client trains its own model on nothing but its own data for 8 epochs, with no aggregation and no communication with anyone else. This baseline shows how much, if anything, federation is actually buying.

E. Evaluation Metrics

For every regime, we report top-1 accuracy (how often the model's top prediction matches the actual next word), top-5 accuracy (whether the correct word shows up anywhere in the top five — closer to how a real suggestion UI would work, offering a few candidates at once), cross-entropy loss, and perplexity. Centralized and FedAvg/DP-FedAvg models are evaluated on the full pooled test set; local-only models are evaluated per-client on their own held-out data, then combined into a test-size-weighted average so the numbers stay comparable to the pooled figures.

VI. RESULTS AND DISCUSSION

Table 2 lays out the final numbers for every regime, and Figures 3 through 5 show convergence, the privacy-utility tradeoff, and final perplexity in turn.

Table 1: Final performance of all training regimes (pooled test set, except local-only which is a test-size-weighted average across 24 clients).

Configuration	Top-1 Acc. (%)	Top-5 Acc. (%)	Perplexity
Centralized (8 epochs, pooled data)	7.23	20.51	294.72
FedAvg (15 rounds, no noise)	4.50	14.84	411.02
DP-FedAvg (σ = 0.01)	4.34	14.69	431.16
DP-FedAvg (σ = 0.05)	4.31	14.82	433.80
DP-FedAvg (σ = 0.10)	4.17	14.36	452.44
Local-only (isolated, weighted avg. of 24 clients)	4.32	15.23	481.66

A. Convergence Behavior

Figure 3 tells a fairly intuitive story. Centralized training converges fastest and climbs highest, from 4.2% to 7.23% top-1 accuracy over 8 epochs, which makes sense given it is optimizing directly over the entire pooled dataset every epoch. FedAvg starts rougher — 2.4% after the first round, reflecting how noisy an average of 24 separately trained clients can be early on — but it climbs steadily and reaches 4.50% by round 15. That is below the centralized ceiling, but it gets there without a single message ever being centralized. DP-FedAvg at $\sigma = 0.05$ stays close to plain FedAvg the whole way through, trailing by roughly 0.2 points of top-1 accuracy at any given round.

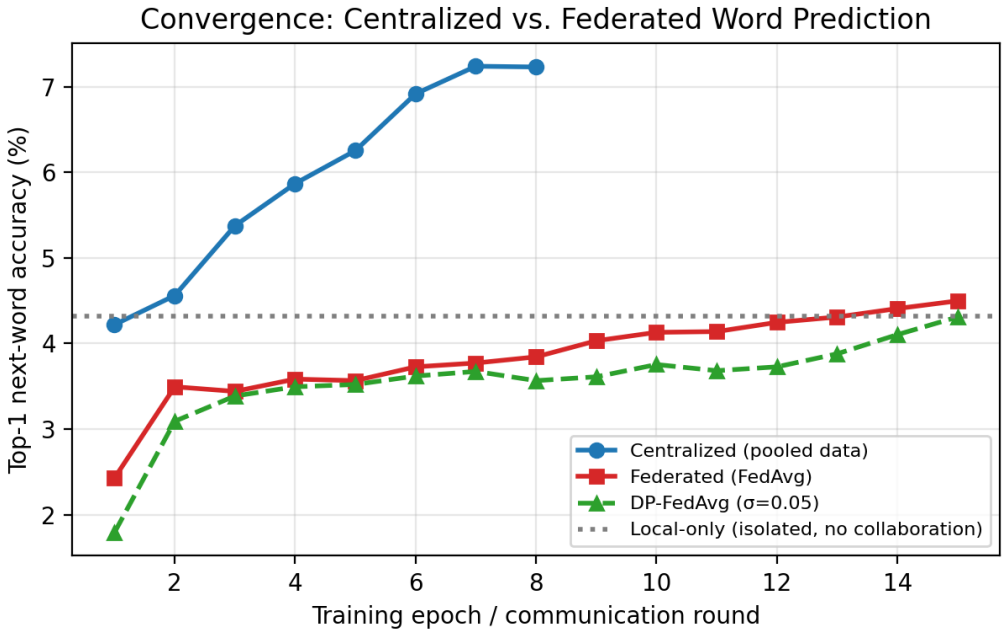

Figure 3: Convergence of top-1 accuracy across training epochs / communication rounds.

B. Privacy-Utility Tradeoff

Figure 4 isolates what the noise itself costs. Pushing σ from 0 up to 0.1 drops top-1 accuracy from 4.50% to 4.17%, and top-5 from 14.84% to 14.36% — a modest, fairly linear decline of around 0.3 to 0.5 points across the range tested. That tracks with what McMahan et al. [4] found: differentially private FedAvg does not cost much, especially once the client population is large. Ours is nowhere near that scale — 24 clients versus the thousands or millions a production system would see — so if anything, the relative cost of a given noise level would be expected to shrink further as more clients get folded into the average.

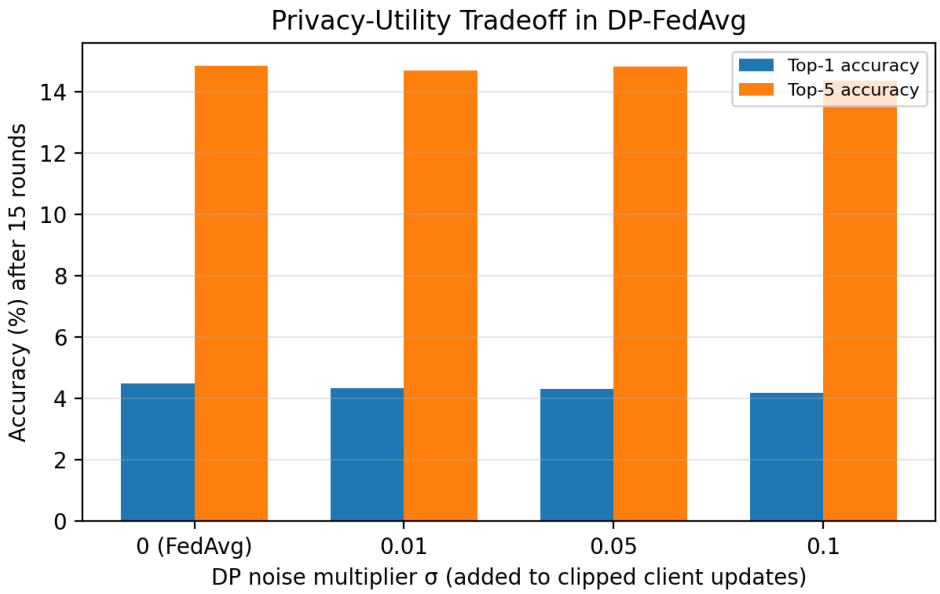

Figure 4: Privacy-utility tradeoff — final accuracy vs. DP noise multiplier σ.

C. FedAvg versus Local-Only Training

One result stood out as more surprising. FedAvg's final numbers (4.50% top-1, 14.84% top-5) sit close to — and on top-5 accuracy and perplexity, slightly behind — what fully isolated local training achieves (4.32% top-1, 15.23% top-5, 481.7 perplexity versus FedAvg's 411.0). In other words, once the client data is this heterogeneous — 24 speakers with barely any vocabulary overlap — averaging their model updates together buys surprisingly little over just letting each client train on its own. This is not a new problem in the FL literature [5]: FedAvg's averaging step implicitly assumes some similarity across clients, and that assumption strains as heterogeneity increases. This is better read as a case for combining federation with some form of personalization — brief per-client fine-tuning after each round, or clustering similar clients before aggregating — than as a weakness of the privacy mechanism itself; we return to this in Section VIII.

D. Perplexity

Figure 5 shows final perplexity across all six configurations. Centralized comes out lowest (best) at 294.7, FedAvg follows at 411.0, the DP-FedAvg variants land between 431 and 452, and local-only training has the worst perplexity of the group at 481.7. That last point is consistent with the top-5 comparison above: isolated local models can hold their own on the coarser accuracy metrics, but they end up with a less well-calibrated probability distribution over the full vocabulary than a model that has had even indirect exposure to other clients through FedAvg.

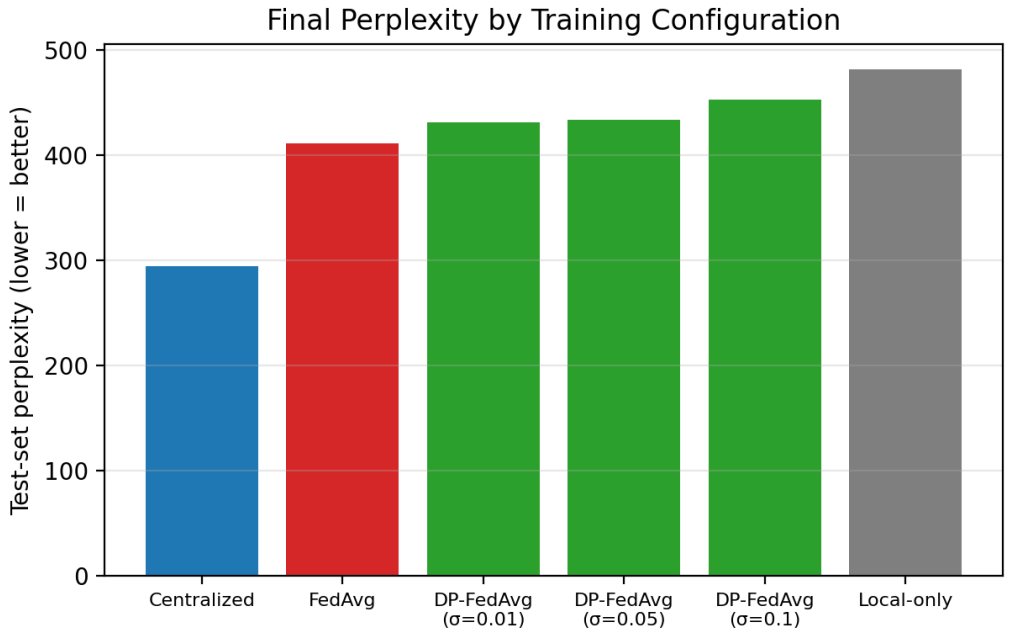

Figure 5: Final test-set perplexity by training configuration (lower is better).

E. Communication Cost and Deployment Practicality

At 200,208 parameters — roughly 0.76 MB per round — the model here fits comfortably within the bandwidth practices production federated keyboards already use, restricting training to devices that are idle, charging, and on an unmetered connection [3]. Moving to a more capable recurrent architecture, like the CIFG-LSTM used in [1], would grow that footprint only slightly, since those architectures are specifically built to keep the per-cell parameter count down for this exact reason.

VII. Limitations

A few limitations are worth stating plainly. The corpus used here is literary dialogue, not real chat data — its speaker-turn structure gives a reasonable stand-in for non-IID per-user text, but real conversations differ in register, length, and topic drift, and these results need revalidating against real, opt-in, anonymized chat logs before the specific numbers should be read too literally. Twenty-four clients is also a small simulation next to a production deployment with thousands or millions of devices, and that gap affects both the achievable accuracy and how strong the privacy-utility tradeoff appears. The feed-forward model used here was chosen to isolate FedAvg/DP-FedAvg dynamics clearly, not to maximize accuracy — a real system would likely reach for a compact recurrent or transformer architecture instead. More importantly, from a privacy standpoint: the Gaussian noise added here follows the shape of DP-FedAvg [4], but no formal (ε, δ)-differential-privacy accountant is run across

training rounds. The noise levels tested demonstrate the clip-and-noise pattern architecturally; they are not a certified privacy guarantee, and a deployable version of this system would need proper accounting — a Rényi-DP or moments accountant — before any specific privacy claim could be made. Finally, cryptographic secure aggregation was not implemented, which would add a further layer of protection by keeping even the server from seeing any individual client's update in the clear.

VIII. CONCLUSION AND FUTURE WORK

This paper set out to replace the centralized training pipeline behind a MERN-stack chat application's word-suggestion feature [8] with something that does not require collecting user text in the first place. The DP-FedAvg design built here keeps raw text on-device throughout, at a real but manageable cost: 4.31–4.50% top-1 accuracy under federation versus 7.23% centralized, with only 0.2–0.5 points of further loss from the privacy noise itself. A second, less expected finding — that plain FedAvg gains little over fully isolated local training once clients are this heterogeneous — points toward personalization as the more interesting open problem here, rather than a shortcoming of the privacy mechanism.

Several directions follow from this. The most obvious is validating the approach on real, opt-in, anonymized chat data rather than literary dialogue. Scaling the client simulation into the thousands would bring the setup closer to production conditions and let the privacy-utility curve be characterized more precisely. Swapping the feed-forward model for a compact recurrent or transformer architecture suited to on-device inference is a natural next step, as is implementing a genuine differential-privacy accountant so the system can make a certified (ε, δ) claim instead of an architectural approximation. Adding cryptographic secure aggregation would close the gap where the server currently sees each client's update, even briefly, before noise is applied. And on the accuracy side, exploring personalization — per-client fine-tuning after aggregation, or clustering clients by writing style before averaging — looks like the most promising way to close the gap between FedAvg and local-only training observed under high heterogeneity. Taken together, these are the steps that would move this from a controlled academic exercise toward something that could plausibly ship inside a real messaging product.

REFERENCES

1. A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated Learning for Mobile Keyboard Prediction," arXiv preprint arXiv:1811.03604, 2018.
2. H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in Proc. 20th Int. Conf. Artificial Intelligence and Statistics (AISTATS), PMLR vol. 54, pp. 1273–1282, 2017.
3. T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied Federated Learning: Improving Google Keyboard Query Suggestions," arXiv preprint arXiv:1812.02903, 2018.
4. H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning Differentially Private Recurrent Language Models," in Proc. 6th Int. Conf. on Learning Representations (ICLR), 2018.
5. P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, et al., "Advances and Open Problems in Federated Learning," Foundations and Trends in Machine Learning, vol. 14, no. 1–2, pp. 1–210, 2021.
6. M. Suliman and D. Leith, "Two Models are Better than One: Federated Learning Is Not Private For Google GBoard Next Word Prediction," arXiv preprint arXiv:2210.16947, 2022.
7. J. Xu, B. S. Glicksberg, C. Su, et al., "Federated Learning for Healthcare Informatics," Journal of Healthcare Informatics Research, vol. 5, no. 1, pp. 1–19, 2021.
8. A. Raj and N. Saraswat, "AI-Enabled Real-Time Chat Application With Intelligent Word Prediction Using MERN Stack And TensorFlow," International Journal of Creative Research Thoughts (IJCRT), vol. 13, no. 5, pp. i134–i145, May 2025.
9. A. Karpathy, "Tiny Shakespeare dataset," char-rnn repository, GitHub, 2015. [Online]. Available: https://github.com/karpathy/char-rnn