REVIEW ARTICLE

Federated Fine-Tuning of Large Language Models on Resource-Constrained Clients: Technical Approaches, Resource Costs, and Applicability

Yunheng Shen[1,2], **Xiao Liu**[3], **Yang Yang**[4,*] **and Hairong Lv**[1,*]

[1] Department of Automation, Tsinghua University, Beijing 100084, China
[2] JD.com, Inc., Beijing 101111, China
[3] School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China
[4] School of Computing and Information Systems, Singapore Management University, Singapore 188065, Singapore

Abstract

Federated fine-tuning adapts language models to distributed data and is widely adopted as a privacy-preserving alternative to centralized training, yet constrained clients must still store model weights and training states, execute updates, and communicate with a server. This review examines four composable routes—parameter-efficient and quantized adaptation, backpropagation-free adaptation, proxy or submodel adaptation, and split federated adaptation—through a common framework that traces the objects each endpoint retains, exchanges, and discloses. When a client retains the complete base, reducing adapter state leaves a base-storage floor; boundary communication depends on input dimensions, sequence length, and interaction frequency, so parameter count alone is insufficient. Keeping data local constrains the raw dataset but not the transmitted object: adapter updates, boundary activations, logits, and cached embeddings expose different surfaces, and the encryption, secure-aggregation, or noise mechanisms that protect them add to the same budgets. Server weight sharing removes duplication but retains private states and caches, and combining routes requires recalculating these costs for the resulting protocol. The analysis identifies where savings arise, where costs move, what each route discloses, and which training, deployment, and trustworthiness conditions govern its applicability.

Keywords: federated learning, large language models, privacy-preserving computation, trustworthy AI, parameter-efficient fine-tuning, zeroth-order optimization, proxy models, split learning, secure aggregation, resource accounting.

Submitted: 27 July 2026
Revised: 08 September 2026
Accepted: 15 September 2026
Published: 20 September 2026

Vol. 2, **No.** 3, 2026.
10.62762/JRSC.2026.225854

***Corresponding author:**
✉ Yang Yang
yang.yang.research@gmail.com
✉ Hairong Lv
lvhairong@tsinghua.edu.cn

Citation
Shen, Y., Liu, X., Yang, Y., & Lv, H. (2026). Federated Fine-Tuning of Large Language Models on Resource-Constrained Clients: Technical Approaches, Resource Costs, and Applicability. *Journal of Reliable and Secure Computing*, 2(3), 194–211.

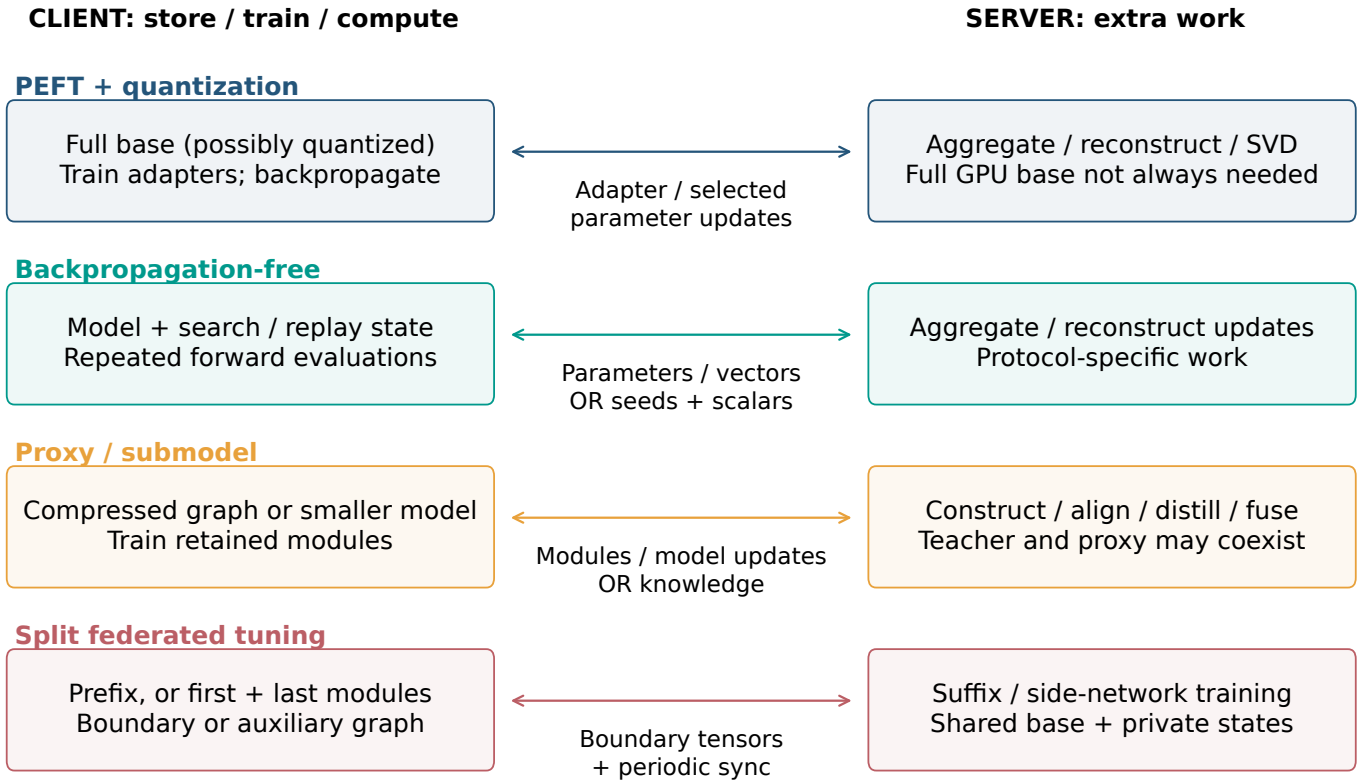

Figure 1. Composable resource-reduction mechanisms. Boxes describe logical work and residency; actual message frequency, initialization, and deployment requirements are method-specific.

1 Introduction

Federated fine-tuning lets data owners adapt a pretrained language model without pooling their training datasets, a central reason it is adopted where privacy, regulatory, or trust requirements prohibit centralization. Keeping data local, however, neither guarantees that every object a client transmits is protected nor makes local training inexpensive. A client may retain the full pretrained weights alongside trainable modules, gradients, optimizer states, saved activations, and temporary buffers. Even with a small trainable module, the total client memory requirement can exceed the available budget. Reducing local storage can also introduce more frequent communication or transfer computation to the server.

Four routes address these constraints: PEFT and quantization, backpropagation-free optimization, proxy or submodel adaptation, and split federated fine-tuning. Their mechanisms can coexist: a low-rank module may be trained on a quantized proxy, or an auxiliary model may support zeroth-order (ZO) learning alongside server-side execution. The difficulty is deciding what their reported savings describe. Trainable parameter count, weight payload, communication per round, and runtime memory measure different costs. Even familiar protocol terms can hide where those costs arise. A black-box interface may execute locally, and a method's "split" may divide local computation without moving it to another device. Figure 1 summarizes the client and server roles of these routes.

Existing surveys cover efficient federated foundation models [1], framework comparisons [2], federated fine-tuning [3, 7], model-access taxonomies [4], and federated PEFT [5, 6]. Split-LLM surveys also examine server memory, scheduling, compression, and joint optimization [8]. Building on this coverage, we follow the objects each route stores, computes, and transmits to determine the conditions under which its resource savings apply.

The review makes four contributions. It separates model access from physical residency when mapping the four routes to endpoint weights, training states, computation paths, and messages. A common accounting framework relates parameter representations, derivative paths, and message schedules to client and server costs. The same

Table 1. Closest prior coverage and the specific accounting emphasis of this review. The comparison concerns scope and accounting boundaries.

Prior work	Established coverage	Emphasis retained here
Framework comparison [2]	Parameter, knowledge, and split frameworks; communication and client computation in §§II–III; a case study in §V.	Separate transmitted objects and workload; retain task-specific evidence.
FedLLM survey [3]	Memory and computation analysis in §§3.3–3.4; zeroth-order, splitting, and compression in §4.5.	Distinguish static state, unknown dynamics, model residency, and parameter-update dimension.
Split-LLM survey [8]	Server memory and scheduling in §4.2.2; communication in §4.3; joint optimization in §4.6.	Extend shared/private/queued-object accounting across the four routes and qualify composition evidence.

object-level view distinguishes the surfaces each route discloses—updates, activations, logits, embeddings, and caches—so that privacy protection and reliability are weighed within the same client, network, and server budgets rather than treated as separate afterthoughts. Applying the framework to implemented combinations then shows how their costs change and which deployment conditions they require. Table 1 relates this emphasis to the closest prior coverage.

Section 2 defines the scope and resource framework. Sections 3–6 examine the four routes, following their mechanisms through training and deployment. Section 7 compares resource costs, compositions, and conditional choices; Section 8 examines the objects each route discloses, the cost of protecting them, and the reliability implications of each schedule; Section 9 presents open challenges and conclusions.

2 Scope and Resource Accounting Framework

2.1 Scope and Literature Selection

We began with a seed bibliography and supplemented it with arXiv title and abstract searches and targeted checks of publisher, conference, and author-maintained pages. The collection covers work first released between January 2021 and September 8, 2026. The archived arXiv results contain 606 entries across seven pages, including material unrelated to this review; this is the number of retrieved records, not the number of studies included.

We focused on English-language original studies in which multiple data owners adapt a pretrained language model, its modules, or its output rule, with a method that addresses client memory, computation, communication, or server costs. Both peer-reviewed papers and identifiable preprints were considered. We excluded work on pretraining, inference alone, preference optimization, and image-only evaluations, as well as reports whose relevant mechanisms could not be verified. Surveys and centralized or two-party studies were used to explain the background and underlying techniques, not as evidence of multi-client federated operation. Findings from encoder-language tasks are discussed within their original model and task settings.

Abstracts helped us judge relevance; for the methods discussed in detail, we consulted the original passages describing the protocol and supporting results. We chose representative studies to show how the four routes differ in what clients store, how updates are computed and exchanged, what the server does, and what can be deployed after training. Selection did not depend on the size of reported performance gains. This approach yields a focused review rather than an exhaustive systematic review: not every retrieved candidate was examined in depth, and unexamined candidates are not presented as full-text exclusions. We compare task findings qualitatively and resource costs under the conditions of each protocol. Different versions of a work are treated as one study, while the version used for analysis is recorded separately. Publication details were checked again on September 15, 2026, without extending the collection period. Withdrawn reports were excluded.

2.2 Notation and Accounting Boundaries

Table 2 defines the principal symbols. Storage cost depends on both the number of parameters and their numerical representation and location. Model-access permission is a separate property: a local black-box interface may still require a locally resident model. Resource expressions specify whether weights and training states remain resident. For sharded, offloaded, or time-varying objects, the accounting follows their actual locations and associated transfers.

Table 2. Principal notation. Parameter subsets and memory objects are disjoint unless explicitly stated.

Symbol	Definition
P; P_f, P_t	Total base-model parameters; resident frozen and mutable parameters in the generic memory model, including adapters where present.
q; q_f, q_t	Adaptation parameters, including any additional modules; frozen and trainable subsets, counted separately from base weights.
P_q, P_{other}; k, g	Quantized and other base-weight subsets; quantization bits per weight and group size.
b_f, b_t, b_g, b_o, b_m	Bytes per frozen weight, mutable weight, gradient, optimizer scalar, and additional master weight.
G, O, P_m; κ	Stored gradient, optimizer, and master-weight scalar counts; combined bytes per trainable parameter and its training states.
$b_{\text{other}}, b_{\text{af}}, b_{\text{meta}}$	Bytes per other base weight, frozen adapter weight, and quantization group metadata.
$b_\uparrow, b_\downarrow, b_c$	Uplink/downlink parameter widths; common width when both directions use the same encoding.
$q_{g\to i}, q_i$	Parameter counts sent to client i and uploaded by that client in one synchronization, respectively.
$b_a, b_{\partial a}$; d_c	Activation and boundary-gradient widths (bytes per scalar); hidden width at a split boundary.
B, S, T, U, a	Microbatch size, padded sequence length, microbatches per round, optimizer updates per round, and accumulation steps; $T = aU$.
d, h, V, r, D	Hidden width, attention heads, vocabulary size, LoRA rank, and random-direction count.
A_i, W_i, Q_i, H_i	Retained activations, workspaces, other dynamic allocations, and their simultaneous maximum for client i.

Memory is reported separately for accelerator, host, and persistent storage, with allocations and logical payloads expressed in bytes. We distinguish static resident storage from simultaneous peak memory, logical message payload from aggregate network traffic, and arithmetic work from execution time. One multiply-add counts as two FLOPs. Transport framing, retransmission, protection, and device-specific kernel behavior require implementation-specific accounting beyond logical payload and arithmetic.

Preparation, adaptation, and deployment can impose different resource requirements. We count model construction, download, and preprocessing separately from local execution and synchronization, then identify the objects needed to deploy the learned result. Parameters, gradients, optimizer states, activations, buffers, and caches are counted once. Recomputation, replay, updates, and encoding are added only when absent from the principal operator count.

2.3 Client Memory and Computation

Let P_f and P_t count disjoint resident frozen and mutable parameters, G the number of stored gradient scalars, O optimizer-state scalars, and P_m additional master-weight scalars. Their element widths are b_f, b_t, b_g, b_o, b_m. Persistent metadata occupy M_{meta}. When these objects remain resident, their static allocation is

$$M_i^{\text{static}} = b_f P_f + b_t P_t + b_g G + b_o O + b_m P_m + M_{\text{meta}}. \tag{1}$$

Total simultaneous memory is $M_i = M_i^{\text{static}} + H_i$, where $H_i = \max_\tau [A_i(\tau) + W_i(\tau) + Q_i(\tau)]$ counts retained activations, workspaces, and other dynamic allocations without overlap. If weight or optimizer-state residency changes over time, those terms must also be included inside the same maximum. Summing maxima observed at different times can overestimate the simultaneous peak.

For FP32 trainable weights, gradients, and two Adam moments, with no extra master copy, the combined weight-and-training-state coefficient is $\kappa = 16$ bytes per trainable parameter. This coefficient depends on the optimizer and precision configuration. Persistent Adam moments are indexed by parameters, so microbatch size does not multiply their storage. A reduced-coordinate or zeroth-order method need not satisfy $G = P_t$ or $O = 2P_t$.

Let q count the adaptation parameters, including any additional modules. For LoRA inserted into matrices $j \in \mathcal{J}$, this gives

$$q = \sum_{j \in \mathcal{J}} r_j (d_{\text{in},j} + d_{\text{out},j}) + q_{\text{extra}}. \tag{2}$$

Here r_j, $d_{\text{in},j}$, and $d_{\text{out},j}$ are the rank and input/output dimensions of matrix j. Additional heads, biases, or embeddings belong in q_{extra}. A frozen full base plus entirely trainable added modules costs $b_f P + \kappa q$ statically under the stated optimizer. If adaptation instead trains existing base weights, Equation (1) moves those weights from the frozen to the mutable allocation; they must not be counted in both. Frozen adaptation parameters still occupy memory. Reducing rank or freezing adapters retains the base and any intermediate activations needed by the remaining training graph [9, 10].

For a declared group-quantization layout, frozen quantized weights contribute

$$M_{\text{weights}} = \frac{k}{8} P_q + b_{\text{other}} P_{\text{other}} + \left\lceil \frac{P_q}{g} \right\rceil b_{\text{meta}}. \tag{3}$$

Here P_q excludes nonquantized weights, g is group size, and b_{meta} is metadata per group. Dequantization buffers are counted in the dynamic term. This group layout differs from QLoRA's nested quantization and paged-optimizer implementation [10].

Budget screening requires

$$M_i^{\text{static}} + H_i \le M_i^{\text{budget}}. \tag{4}$$

A configuration is excluded if its static allocation exceeds the budget under the stated residency assumptions. Otherwise, the remaining headroom sets the permissible bound on H_i. Activation storage depends on graph structure, earliest trainable position, attention implementation, and recomputation. Materializing attention scores can require BhS^2 elements; implementations that avoid this tensor require a different count. Activation storage must follow the differentiated graph, not just the fraction of trainable parameters.

For operation o, let n_o count executions and F_o denote its FLOPs. Client arithmetic is accounted for as

$$F_i = \sum_o n_o F_o + F_{\text{update}} + F_{\text{replay}} + F_{\text{recompute}} + F_{\text{encode}}. \tag{5}$$

A linear forward pass costs approximately $2BSd_{\text{in}}d_{\text{out}}$ FLOPs; input and weight derivatives each require a comparable matrix multiplication. Freezing a weight removes its weight derivative, but not an input derivative needed by an earlier trainable module. Two-sided finite differences with D directions require $2D$ graph evaluations, plus perturbation and update work. This rule does not apply unchanged to analytic directional derivatives, evolutionary search, or hybrid methods.

Server arithmetic is decomposed separately:

$$\begin{aligned} F_s = {} & F_{\text{agg}} + F_{\text{construct}} + F_{\text{align/distill}} \\ & + F_{\text{remote training}} + F_{\text{replay}} + F_{\text{encode}}. \end{aligned} \tag{6}$$

Each term is weighted by its execution frequency, with one-time construction counted at initialization. Parameter summation, dense product decomposition, proxy distillation, and suffix training require separate estimates. For an unchanged differentiable graph, partitioning preserves $F_i + F_{s,i} = F_{\text{whole}}$ apart from transport-related operations. Server arithmetic sums the serviced client workloads even when execution is serial; simultaneous memory instead depends on residency and scheduling.

2.4 Communication and Server Resources

Unless stated otherwise, recurrent payloads are counted per communication round. For message type m, let n_m be the number of transmissions in that round, e_m the number of scalars per message, s_m bytes per scalar, and z_m metadata bytes per message. Each direction is counted independently:

$$V_i^{\uparrow/\downarrow} = \sum_{m \in \mathcal{M}^{\uparrow/\downarrow}} n_m (e_m s_m + z_m). \tag{7}$$

Total training payload is obtained by summing the actual per-round payloads and adding one-time initialization or feature-collection messages that are not included in those rounds. Transfers of trained modules for deployment are reported separately. Aggregate network traffic also depends on transport and delivery: unicast sends a copy to each recipient, whereas multicast can share transmissions. For one bidirectional adapter synchronization, the payload is

$$V_i^{\text{param}} = b_{\downarrow} q_{g \to i} + b_{\uparrow} q_i + V_{\text{meta}}. \tag{8}$$

Here $q_{g \to i}$ and q_i count the parameters sent to client i and uploaded by it, respectively; V_{meta} covers that synchronization. The per-round adapter payload sums Equation (8) over the synchronizations actually performed in the round. The familiar $2b_c q$ is the payload of a single synchronization with identical bidirectional objects and precision. Sparse updates require positions or masks. Seed-based protocols require actual seed, coefficient, history, and resynchronization counts. For N_{pub} public inputs of length S_{pub}, vocabulary size V, and logit width b_z bytes, sharing logits gives $N_{\text{pub}} S_{\text{pub}} V b_z$ per

complete one-way exchange; top-k_z encoding gives $N_{\text{pub}} S_{\text{pub}} k_z (b_z + b_{\text{idx}})$, where k_z is the retained entry count and b_{idx} the bytes per vocabulary index, plus losses and other metadata.

For split execution, the per-round payload sums all boundary messages and synchronization events within the same round:

$$V_i^{\text{split}} = \sum_c (n_{a,c} b_{a,c} e_{a,c} + n_{g,c} b_{\partial a,c} e_{g,c}) + V_{\text{sync}} + V_{\text{labels/meta}}. \tag{9}$$

At boundary c, $n_{a,c}$ and $n_{g,c}$ count activation and gradient messages in the round, while $e_{a,c}$ and $e_{g,c}$ give their scalar counts; $b_{a,c}$ and $b_{\partial a,c}$ give bytes per scalar. The synchronization and label/metadata terms cover that same round. A single boundary with one same-shaped activation and gradient per microbatch therefore contributes $TBSd_c(b_a + b_{\partial a})$. Multi-boundary and U-shaped graphs require the sum, while decoupled or cached-feature methods can change both message counts and the learning objective. Gradient accumulation does not cancel per-microbatch boundary exchanges.

The server model separates shared objects, resident client-specific states $\mathcal{R}(\tau)$, and in-flight graphs $\mathcal{I}(\tau)$:

$$M_s = \max_\tau \Big[M_{\text{shared}} + \sum_{j \in \mathcal{R}(\tau)} M_{\text{private},j} + \sum_{b \in \mathcal{I}(\tau)} A_b + M_{\text{agg}} + M_{\text{cache}} + M_{\text{runtime}} \Big]. \tag{10}$$

An aggregation-only server may operate without a GPU-resident base. Its working set still depends on whether aggregation uses streaming sums, factor stacks, matrix decomposition, or distillation. Sequential computation can reduce active graphs while leaving many independent adapters or queued activations resident. Physical aliasing must be established before treating two named model objects as one allocation.

3 Parameter-Efficient and Quantized Adaptation

PEFT reduces the optimized state while retaining pretrained computation. In low-rank adaptation (LoRA), a frozen matrix and two trainable factors implement $\mathbf{W}_0 + \mathbf{BA}$ [9]. Dense base-weight gradients and optimizer states are removed, but locally executed base weights remain resident. Frozen layers may also need to propagate gradients to earlier trainable modules. Prompt-based methods change a different object: their additional tokens affect attention state, so their costs cannot be inferred from the LoRA parameter formula.

3.1 Trainable State and Aggregation

Heterogeneous LoRA methods differ in how local factors become a communicated aggregate. HetLoRA truncates distributed factors to client ranks, prunes ranks locally, and aggregates with sparsity-dependent weighting [11]. The backbone remains intact, and the server must compute product norms for weighting as well as combine updates. FLoRA uses factor stacking to represent the weighted sum of client products exactly [12]. That exact aggregate can have a rank that grows with participating client ranks. Its downlink can consequently exceed an individual client's uplink even when each local adapter is small.

FlexLoRA constructs the aggregate in product space and redistributes client-specific ranks through singular value decomposition (SVD) [13]. FraQ recompresses factor stacks in compact coordinates, avoiding a dense full-weight decomposition [14]. Similar client-facing LoRA factors thus conceal different server working sets and arithmetic. Costs can also change between training phases: in FraQ's RoBERTa classification setup, the classifier is trained and aggregated for the first five communication rounds and then frozen [14].

Freezing an adapter factor changes its training state without removing its weights. FFA-LoRA fixes $\mathbf{A}_0$ and trains only $\mathbf{B}$ [15]. With b_{af} bytes per frozen entry, the factor occupies $b_{\text{af}} r d_{\text{in}}$ bytes; trainable states scale with $r d_{\text{out}}$. "Half the trainable parameters" requires matching factor dimensions and describes neither total memory nor FLOPs. SLoRA instead begins with sparse priming before low-rank adaptation [16], adding initialization training, communication, and decomposition. Fed-pilot selects LoRA modules under a client memory constraint [17]. For Equation (1), all resident adapters must be counted, including the frozen subset; the earliest trainable position determines how far gradients propagate.

3.2 Numerical Representation and Applicability

Quantization may target resident weights or transmitted updates, with different consequences. QLoRA combines quantized frozen weights with low-rank training in a centralized setting, including metadata and memory-management techniques [10].

HAFLQ supplies a federated protocol with blockwise base quantization, heterogeneous adapter treatment, and bandwidth-aware quantization of outgoing adapter information [18]. These act on base residency, trainable state, and message encoding, respectively. FedLPP quantizes *proxy LoRA matrices* distributed over a public backbone [19]; the backbone weights are not the object of this quantization saving.

Rank also interacts with aggregation and data heterogeneity. In its PaLM 2-XXS/XS evaluations, HetLoRA reports better Reddit summarization and multi-session-chat results than the homogeneous-rank and reconstruction/SVD comparators, while remaining below full fine-tuning [11]. Larger homogeneous ranks can improve early fitting yet worsen the final result through overfitting in its rank study. These findings support matching rank and aggregation to the evaluated client distribution, rather than treating rank as a monotonic indicator of task quality.

A client must still retain the required base, execute the numerical kernels and backward path, and accommodate dynamic allocations. Rank, injection location, quantization, and aggregation representation can each affect adaptation. The resulting adapter is usually attached to, or merged into, its corresponding base, so deployment retains that model dependency.

4 Backpropagation-Free Adaptation

Backpropagation-free adaptation reduces retained differentiation state but still executes the model and maintains the search or update state its optimizer needs. With in-place two-sided finite differences, a seed can regenerate each perturbation in exchange for repeated forward evaluations [20]. Stored or parallel directions, momentum, and reconstruction buffers change this memory requirement. Equation (1) therefore counts G, O, and auxiliary objects for the specific implementation.

4.1 Estimators, Reconstruction, and Messages

FwdLLM combines PEFT with perturbed inference, server-controlled direction selection, and variance-aware pacing [21]. Its forward-gradient estimator uses numerical differentiation, whose assumptions differ from those of an exact directional derivative; the server also runs a PEFT profiler on public data. The quantized LLaMA configuration demonstrates that these mechanisms can operate together. It does not isolate a general interaction benefit, and comparisons involving different hardware paths cannot attribute runtime differences solely to the estimator.

FedKSeed encodes updates through a seed-based communication protocol [22]. Clients retain a common pretrained model and reconstruct the current model from a finite seed pool and accumulated coefficients. They upload local seed/scalar histories, allowing the server to update accumulators without necessarily holding model weights. The encoded update dimension shrinks at fixed protocol counts, while initial-model delivery, resets, random-number reproducibility, and reconstruction remain part of the protocol cost. DeComFL similarly requires returning clients to recover missed updates, and distinguishes scalar aggregation from a first-order projected-gradient alternative [23]. Zeroth-order communication costs consequently depend on the protocol and the history each returning client needs.

Removing backpropagation does not itself compress messages. FedMeZO can use LoRA while continuing to exchange model or adapter parameters [24]. FedSPZO divides computation into two *local* perturbation blocks with different allocations of directional evaluations; both blocks remain on the client [25]. Its communication and compute account includes scalar uplink, model downlink, and server replay. The split is local to the perturbation scheme; it does not move either block to a server.

4.2 Search Space, Task Performance, and Deployment

Backpropagation-free search also extends beyond finite-difference full-weight updates. FedBPT optimizes a projected prompt through covariance matrix adaptation evolution strategy (CMA-ES) and exchanges search-distribution parameters, including a covariance structure [26]. Covariance state can scale quadratically with search dimension. The black-box interface also leaves physical residency to be specified: a local backend retains its weights, while a remote backend introduces query traffic and data-access conditions.

FedKSeed's seed-cardinality study on its federated instruction-tuning tasks reports diminishing task-performance gains as the candidate pool grows, while importance-based sampling can reach similar results with fewer candidate directions [22]. Expanding the seed pool offers more directions but increases state distribution and reconstruction work. The sampling rule and retained search subspace must

be chosen for the pretrained model and target tasks, alongside the message budget.

Shared randomness also appears in first-order methods. FedKRSO evaluates a first-order gradient with respect to a low-dimensional temporary factor, then updates the full weight matrix and retains compressed moments [27]. Ferret performs first-order local training before projecting updates for communication [28]. Those first-order steps still require their corresponding activations and derivative states. For backpropagation-free adaptation, applicability depends on estimator dimension, direction count, perturbation scale, replay conditions, and the execution interface. Full-weight updates produce a changed base; adapter or prompt search produces modules that still depend on the base or its query interface at deployment.

5 Proxy- and Submodel-Based Adaptation

Proxy and submodel methods reduce the graph a client executes or replace it with a smaller model. The smaller graph may provide updates for retained modules, approximate the missing part of a large model, or exchange knowledge and prediction corrections with another model. These transfer procedures lead to different adaptation products. Offsite-Tuning establishes the centralized/two-party emulator mechanism; subsequent multi-client protocols provide the federated evidence [29].

5.1 Replacing and Aligning the Client Graph

In FedBiOT, "adapter" can refer to retained Transformer blocks as well as low-rank modules [30]. Trailing blocks remain adaptable, preceding blocks are compressed into an emulator, and both receive LoRA modules. Clients train the adaptable blocks' LoRA; the server uses public data to align the emulator with the uncompressed model. The client thus retains frozen emulator-LoRA weights as well as trainable adapter-LoRA state. Initial compressed-model delivery and later emulator updates contribute at different downlink frequencies, while teacher/emulator execution, alignment data, and optimizer states remain server costs.

FedPFT reduces feed-forward network (FFN) neurons while preserving the layerwise graph and aligning the proxy before and during federated tuning [31]. Attention computation remains, so reducing FFN width does not scale all arithmetic by the same fraction. Periodic realignment adds neuron-related updates and computation to adapter exchange. FedPruner takes a different approach, selecting layers or components for client-specific submodels and aggregating updates by original layer identity [32]. Its local similarity profiling adds an earlier workload. For both approaches, construction and profiling need their own residency account: fitting the finished submodel does not establish that the client can construct it within the same budget.

FedSODA combines a pruned emulator, four-bit NormalFloat (NF4) representation, LoRA, and public-data alignment [33]. On BoolQ with LLaMA3-8B, Qwen2-7B, and Mistral-7B, its ablations favor retaining both prealignment and recurrent realignment over omitting either stage [33]. Maintaining correspondence with the full model matters to task performance in these configurations. The combined static allocation comprises quantized weights and metadata, frozen adapter weights, and the trainable adapter subset with its training state. Counting these objects directly avoids crediting pruning, quantization, and PEFT twice for the same reduction.

5.2 Update Transfer, Task Performance, and Deployment

The destination of a proxy update determines what must be deployed. FedProxy builds a shared proxy with public instruction data, coordinates heterogeneous updates, and replaces corresponding full-model layers with updated proxy layers [34]. Structural correspondence permits the final plug-in step without further training, although compression and federated optimization incur training costs earlier. FedPT instead trains a smaller client model and combines large-model logits with the difference between tuned and original small-model logits [35]. This changes the prediction rule while leaving the large-model base weights fixed. Public data support server-side small-model distillation, and inference still needs the models contributing to the rule.

FedMKT uses LoRA and bidirectional knowledge transfer to adapt both the server's large model and the clients' smaller models [36]. Public inputs provide a shared basis for exchange, with token alignment handling tokenizer mismatches. The payload depends on top-k_z logits, indices, loss values, and transfer frequency. Parameter shapes can differ across models, but the vocabulary-dependent payload and alignment work remain. The server also executes models and backpropagates for distillation, exceeding the work of parameter aggregation alone.

Public transfer data are part of the learning design, not just an incidental server allocation. A smaller proxy is useful only insofar as its updates or knowledge remain relevant to the target model. Surrogate-gradient mismatch, poorly covered transfer inputs, tokenizer differences, and changed optimization targets can impair adaptation; the outcome depends on the protocol and data. The deployment account should identify whether training produces a local proxy, a mapped full model, an adapted small model, or a composite prediction rule.

6 Split Federated Adaptation

Split execution distributes a computation graph between clients and a server. In a conventional differentiable split, activation uploads and gradient feedback allow both partitions to update. If the graph, loss, arithmetic, and update schedule are unchanged, this placement preserves the chain rule exactly. Task-performance differences arise when the protocol also changes aggregation, partitions, compression, or scheduling; placement alone does not explain them.

6.1 Coupled Execution and Server Sharing

SplitLoRA combines client-prefix and server-suffix LoRA training with periodic client-adapter aggregation [37]. Clients send labels and activations, which the server combines for shared training. Its training-round unit denotes microbatch-level split work; adapter synchronization follows a separate interval. Equation (9) preserves both frequencies. HSplitLoRA additionally adapts ranks and partition locations and aggregates heterogeneous factors [38]. Its weight-importance estimation and enlarged aggregate factors add costs beyond the partition itself. Rank and profiling must be considered together with the cut point.

Server memory depends on what the implementation shares. The memory-efficient framework of Chen et al. [39] retains one complete frozen server base and switches between client-specific suffix adapters while processing clients sequentially. The server base is counted once, alongside independent adapters, optimizer states, and waiting activations. Client memory follows its own resident partition. SflLLM distinguishes split-training traffic to the main server from parameter synchronization with the federated server, jointly optimizing radio allocation, rank, and partition decisions [40]. The two server roles therefore have separate communication and execution costs.

SplitFT changes cut-layer LoRA ranks and client allocation [41]. Adapter gradients and boundary gradients have different shapes and must be counted separately. A lower LoRA rank does not, without an explicit boundary transform, reduce a $B \times S \times d$ activation or activation gradient. FlexP-SFT provides a different objective: client-specific first/last layers and a shared server cooperate without client-parameter aggregation [42]. Its U-shaped execution and additional alignment messages still incur communication. Clients remain coupled through shared server training and retain their local embedding and output modules.

6.2 Decoupled and Cached Execution

HERON-SFL uses an auxiliary client objective for zeroth-order learning while the server suffix receives first-order updates [43]. In its GPT-2/E2E configuration, client, auxiliary, and server components train LoRA with other weights frozen, so synchronization concerns updated adapters. Auxiliary networks range from Transformer blocks with unembedding to a smaller LayerNorm/unembedding variant. The GPT-2-Medium ablation finds final training loss relatively insensitive to auxiliary capacity for HERON-SFL, while the first-order comparator benefits from a larger auxiliary model. This supports the minimal variant in that setting and makes its smaller residency and forward workload relevant to configuration choice.

One-shot SFT uses weight tying to supervise server activations with target embeddings and A-Loss, whose negatives include a historical embedding cache [44]. With a finite cache, A-Loss approximates full-vocabulary cross-entropy. Online L-shaped training retains split-gradient feedback. One-shot mode fixes the client prefix and buffers activation–target-embedding pairs for the whole local dataset before uploading them. The report does not specify whether this buffer resides in accelerator, host, or persistent storage. For N_i uncompressed, fully retained samples of padded input/target lengths S_x, S_y, its object size is

$$M_{\text{buffer},i} \approx N_i(S_x d_c b_a + S_y d_e b_e), \tag{11}$$

where d_e and b_e denote target-embedding width and bytes per scalar. Variable-length samples require a sum over their stored lengths. The preparation buffer is additional to the server's dataset cache; streaming the upload would constitute a different preparation protocol.

Table 3. Protocol-conditioned resource comparison. H_i has the simultaneous dynamic-memory meaning defined in Section 2.3; communication entries identify message objects, with per-round counts defined in Section 2.4 and one-time preparation/deployment transfers reported separately.

Configuration	Client memory objects	Logical payload	Server work and state
Full-base LoRA	$b_f P + \kappa q + H_i$ for fully trainable added adaptation parameters.	Per synchronization: downlink $b_{\downarrow} q_{g\to i}$; uplink $b_{\uparrow} q_i$; protocol metadata.	Aggregation; stacked factors or decomposition where required.
Quantized proxy with LoRA	Mixed base and adapter states in Eq. (12).	Actual adapter, proxy, and alignment updates at their respective frequencies.	Proxy construction, teacher execution, alignment states and data.
In-place zeroth-order training	Mutable weights, search/replay state, and forward workspace.	Seeds, coefficients, history, or parameters according to the protocol.	Scalar aggregation or model reconstruction and replay.
Mutual knowledge transfer	Local small model, adapters, public-input evaluation state.	Logits, indices, losses, and alignment information.	Large-model adaptation, token alignment, public-data execution.
Differentiable split	Client partition, training state, and waiting graphs.	All boundary tensors plus periodic parameter synchronization.	Suffix, shared/private states, active graphs, and queues.
Cached-feature training	Frozen graph and preparation buffer; Eq. (11) for retained pairs.	Feature collection and final modules returned for deployment.	Dataset cache, suffix or side-network training; loss-specific cache.

Fed MobiLLM extracts selected multilayer features and prediction residuals from frozen heterogeneous client backbones [45]. The server caches them and trains a shared side-network with model-specific projections. Clients later download that network and their projections for local inference. Their inference footprint therefore differs from the footprint during feature extraction, while optimization and cache storage reside on the server.

6.3 Task Performance and Deployment Conditions

On the evaluated WikiText-2 and MMLU settings, online L-shaped training reports task performance comparable to its U-shaped comparator [44]. The comparison concerns online adaptation. In one-shot mode, removing recurrent feedback fixes the feature map used for suffix optimization, so the same quality conclusion does not follow. A coupled split preserves the original graph's learning capacity under matched losses and update schedules; compression, auxiliary losses, and frozen features introduce separate approximation or optimization choices. Deployment then requires either recombining the trained partitions, sustaining distributed execution, or installing the returned modules specified by the protocol.

7 Cross-Route Analysis and Compositions

7.1 Comparison under Common Resource Boundaries

The resource advantage of a route depends on which object limits participation. PEFT reduces training state, quantization changes selected representations, backpropagation-free methods replace derivative requirements, proxies change the local graph, and splitting moves execution between endpoints. Table 3 uses the boundaries of Section 2 to compare these changes. A small update payload can still require a large client-resident base; reducing client computation can also leave substantial training work on the server.

For a quantized proxy with mixed base representations and frozen/trainable adapter subsets, the fixed-residency client model is

$$M_i = \frac{k}{8} P_q + b_{\text{other}} P_{\text{other}} + b_{\text{af}} q_f + \kappa q_t + M_{\text{meta}} + H_i. \tag{12}$$

The base subsets P_q and P_{other} are disjoint, as are adaptation subsets q_f and q_t; any parameters assigned to the latter are excluded from the base subsets. Frozen adaptation parameters contribute weights only; κq_t includes trainable weights and their states. The FP32/Adam configuration uses $\kappa = 16$ without

Table 4. Composition mechanisms and evidence boundaries. An implementation establishes that components operated together; an ablation supports only the component and comparison it isolates.

Method / configuration	Combined mechanisms and affected objects	Added costs or conditions	Evidence and boundary
HAFLQ [18]	Quantized base + heterogeneous LoRA + encoded updates: stored weights, training states, and messages.	Metadata, nonquantized states, encoding and dequantization.	Federated implementation; effects depend on the selected precision and rank.
FwdLLM [21]	Perturbed inference + PEFT + quantized LLaMA: derivative path, trainable space, base representation.	Direction evaluations, server profiling, feedback.	Federated configuration; complete-method results do not isolate universal interaction gains.
FedBiOT [30] FedSODA [33]	Proxy + LoRA + alignment; FedSODA also uses NF4.	Frozen emulator adapters, teacher execution, transfer data and updates.	Federated implementations; FedSODA ablates alignment stages in its tested setup.
SplitLoRA / HSplitLoRA [37, 38]	Split + LoRA: placement of base and training state.	Boundary tensors, synchronization, server states; heterogeneous-rank aggregation.	Federated implementations with method-specific schedules and ranks.
HERON-SFL (language models) [43]	Split + LoRA + local ZO + auxiliary model.	Auxiliary graph, periodic features, adapter synchronization.	Federated implementation; auxiliary-capacity ablation in GPT-2 configurations.
QLoRA [10]	Quantized base + LoRA + paged optimizer.	Metadata, numerical kernels, memory transfers.	Centralized foundation; federated protocol costs require separate evidence.
Quantized proxy + ZO + split	Conditional composition of graph, state, and placement changes.	Actual auxiliary graph, remote evaluations, alignment and messages.	Mechanism inference; resource feasibility requires the resulting object and workload counts.

adding trainable weights twice. The simpler uniformly quantized proxy expression is the special case $P_{\text{other}} = q_f = 0$.

The expressions in Table 3 distinguish costs that respond to different design choices. With a fully resident frozen base, reducing adapter rank changes the trainable state but leaves the base-storage floor. Quantization changes the representation of selected weights; pruning changes the retained graph. Their benefit depends on which object dominates the client's memory, including dynamic allocations required by the remaining training path.

Parameter exchange and split-boundary exchange also follow different scaling rules. Adapter messages depend on the communicated parameter count and encoding, whereas boundary tensors depend on microbatch size, sequence length, hidden width, and interaction frequency. A smaller adapter does not shrink a boundary tensor unless the protocol explicitly transforms that tensor. Initialization, periodic synchronization, and cached-feature collection must be included at their own frequencies.

Sharing frozen server weights reduces duplication while retaining client-specific adapters, optimizer states, active graphs, and queued or cached representations. Sequential scheduling can reduce simultaneous computation graphs without removing persistent client states. Moving a partition to the server therefore changes both endpoint budgets; client savings alone do not establish the feasibility of the complete protocol.

7.2 Composition Mechanisms and Evidence

Complementary components can reduce different limiting objects, but their costs must be counted together. Quantizing a frozen proxy changes bytes per retained base weight, while pruning changes the number of retained weights and LoRA changes the trainable subset. Their combined memory follows Equation (12). After PEFT has restricted gradient storage to adapters, a zeroth-order component can remove that remaining gradient allocation and alter saved activations; it cannot claim a second reduction of dense gradients already absent from the PEFT graph.

Table 5. Advantages, limitations, and suitable application scenarios of the four technical routes. Scenarios describe resource and deployment conditions, not measured superiority across tasks.

Technical route	Main advantages	Main limitations	Suitable application scenarios
PEFT and quantization (Section 3)	Reduces trainable state and adapter payload; base quantization also lowers resident weight storage.	The required base must still fit. Backward activations, numerical kernels, and heterogeneous-rank aggregation can remain costly.	Clients can host the base at the chosen precision but have limited training-state memory or bandwidth; periodic adapter exchange supports local adaptation.
Backpropagation-free (Section 4)	Avoids reverse-mode activation retention; seed/scalar protocols can additionally reduce update traffic.	Repeated forward evaluations, estimator variance, and replay can be costly. Base residency remains; smaller messages require an explicit encoding protocol.	The model fits for inference but backward state does not, or only a suitable black-box interface is available; extra evaluations are affordable.
Proxy or submodel (Section 5)	A smaller local graph lowers resident weights and execution cost; suitable transfer protocols can accommodate heterogeneous models.	Transfer mismatch, construction and alignment costs, and public-data requirements can limit use. The final product may require more than the client proxy.	Clients cannot host the full target model, but a smaller model fits and a server can support compatible update transfer or knowledge alignment.
Split federated (Section 6)	Moves part of the graph and training work to the server; frozen server weights can be shared across clients.	Coupled execution needs frequent boundary exchange and server capacity. Cached variants add storage and restrict adaptation through fixed features.	Constrained clients have a reliable low-latency server link; for intermittent links, cached-feature variants suit tasks that tolerate fixed client representations.

Table 4 distinguishes implemented federated combinations, component-specific ablations, centralized foundations, and mechanism-based extensions. For example, FedSODA's alignment ablations support its prealignment and recurrent realignment stages within the tested setup [33]. They do not isolate independent gains for every compression component. Configuration choices can use this component-specific evidence alongside the combined protocol account; separately reported saving percentages are not additive.

7.3 Conditional Selection Guidelines

The dominant allocation provides the first selection criterion. When frozen base weights already exceed a client's memory budget, reducing adapter rank cannot admit that client. Weight compression, a smaller graph, offloading, or partitioning must change the resident base. When the base fits but backward state does not, trainable placement, numerical state, the derivative method, and server execution offer different ways to reduce the excess. In either case, the static margin is the allowance available for activations and runtime allocations.

Connectivity constrains the frequency of interaction. Parameter and seed protocols can synchronize after several local updates; conventional split execution introduces dependencies at each microbatch. Long sequences and multiple boundaries increase the exchanged tensors, while short local compute intervals leave frequent opportunities to wait for the network. A low-latency local link and an intermittent bandwidth-limited link therefore suit different schedules. Cached-feature training reduces online dependence by fixing representations for a server-side objective, at the cost of storage and a changed adaptation procedure.

Client savings must fit the available server capacity. A parameter aggregator can work without an accelerator-resident base, but a distillation server must execute the teacher or student required by its objective. Split systems can share frozen weights while retaining independent optimizer states, graphs, and queues. Moving learning off the client may transfer feature storage as well as optimization to the server. Client memory and arithmetic, bidirectional payload and its network realization, and server memory and arithmetic consequently impose separate admission

constraints.

Deployment can rule out an otherwise feasible training configuration. Adapters need their corresponding bases, proxy updates need a transfer map, and split models need colocation or continued remote execution. FedPT retains a multi-model output rule; FedMKT adapts both model scales; Fed MobiLLM returns additional client modules after feature extraction. Public-data alignment and remote execution must also fit the application's data and model-access policies.

Data locality alone does not establish privacy. Updates, activations, label-related residuals, target embeddings, and logits are different disclosed objects. Their security properties depend on the threat model and the protection mechanism, whose resource costs are examined in Section 8, which develops this disclosure surface route by route and relates it to the resource budgets above.

7.4 Summary of Route Applicability

Table 5 summarizes the main advantages, limitations, and suitable application scenarios of the four routes. These are conditional choices rather than a performance ranking: deployment must satisfy the client, network, and server budgets together. Routes can be combined when the resulting protocol preserves the required adaptation and deployment behavior.

8 Security, Privacy, and Reliability Considerations

Federated fine-tuning is adopted as a privacy-preserving alternative to centralized training: raw examples remain with their owner, and only derived objects are exchanged. Within a reliable and secure computing setting, this data-locality property is a starting condition rather than a guarantee. What an honest-but-curious server or a network observer can attempt to reconstruct is set by the object a client transmits, not by the fact that its dataset stays local. Because the four routes transmit different objects, they expose different surfaces and admit different protections, each with its own resource cost that adds to the budgets of Section 2.

8.1 Disclosed Objects by Route

Table 6 lists the objects each route reveals during training and after deployment. Parameter-efficient and quantized adaptation exchanges adapter updates and aggregates, so the disclosed object is the low-rank or quantized update rather than the frozen base. Several protocols in this route already treat the update itself as the privacy-relevant object: FFA-LoRA fixes one factor to change what is optimized and transmitted [15], FedLPP quantizes the distributed proxy LoRA matrices [19], and HAFLQ applies bandwidth-aware quantization to outgoing adapter information [18]. Backpropagation-free adaptation can replace parameter messages with seed and scalar histories [22, 23], but a returning client must still recover the history needed to reconstruct the current model, and a black-box interface backed by a remote server introduces query traffic and data-access conditions rather than removing them [26]. Proxy and submodel methods disclose logits, indices, and losses over public inputs [36], or a composite output rule that combines large- and small-model predictions [35]; the public transfer data are part of the learning design, not an incidental allocation. Split execution exposes boundary activations, gradients, and labels [37], and cached variants additionally retain activation–target-embedding pairs or extracted features and residuals [44, 45], objects that persist beyond a single round.

8.2 Protection Mechanisms and Their Costs

Whatever the disclosed object, its security property depends on the threat model and the protection applied. Encryption, secure aggregation, and additive noise each change the resource account: they enlarge messages, add server or client computation, or perturb the very updates whose small size motivated the route. A protection that suits one disclosed object need not suit another. Masking an aggregated adapter update differs from protecting the per-microbatch activations that a split server must read in the clear to continue its forward pass, and both differ from bounding what a shared logit or a cached embedding reveals. The accounting framework extends naturally to this concern: a route is admissible only when its disclosed objects can be protected to the required threat model within the same client, network, and server budgets, not merely when its unprotected messages fit.

8.3 Reliability and Dependable Operation

Reliability constraints act on the same objects. Seed- and coefficient-based protocols depend on reproducible random-number generation and on returning clients recovering missed updates, so a lost or reordered message becomes a correctness concern rather than only a performance one [22, 23]. Split execution couples client and server within a round, so an interrupted boundary exchange stalls both

Table 6. Objects disclosed by each route and the resource effect of protecting them. Entries follow the mechanisms described in Sections 3–6; they characterize the disclosure surface rather than measuring any specific attack.

Route	Disclosed objects during training	Persisted or deployed objects	Protection and its resource effect
PEFT and quantization	Adapter updates and aggregates, low-rank or quantized.	Adapter attached to or merged into its corresponding base.	Secure aggregation and encryption enlarge messages; additive noise perturbs the small update that motivated the route.
Backpropagation-free	Seed and scalar histories, or model/adapter parameters; search-distribution parameters.	Changed base, or modules that still depend on the base or its query interface.	Reproducibility and history recovery are correctness conditions; a remote black-box backend adds query traffic and access conditions.
Proxy or submodel	Logits, indices, and losses over public inputs; composite output rules.	Local proxy, mapped full model, adapted small model, or composite prediction rule.	Public transfer data are part of the design; alignment and distillation add server execution beyond aggregation.
Split federated	Boundary activations and gradients, and labels; cached activation–embedding pairs or features and residuals.	Recombined partitions, sustained distributed execution, or returned modules.	The server consumes activations in the clear to continue the forward pass; caches persist across rounds and enlarge the protected surface.

partitions, whereas parameter and cached-feature protocols tolerate intermittent participation by synchronizing less often. A server that shares one frozen base across sequentially scheduled clients concentrates availability on a single resident model [39]. A dependable deployment therefore weighs not only the resource budgets but the failure modes each schedule introduces, and cached-feature variants trade online coupling for stored state that must itself remain available and protected.

9 Open Challenges and Conclusions

9.1 Limitations

The accounting specifies resource objects and workloads, leaving dynamic allocations dependent on trainable placement, attention kernels, recomputation, memory hierarchy, and scheduling. Architectural differences also change the allocations and computation required at each endpoint. Logical payload and dominant FLOPs describe messages and arithmetic. Network traffic, elapsed time, and energy additionally require an implementation model.

The selected reports differ in model families, tasks, data partitions, and comparison budgets. Fixed source versions and targeted primary-text inspection anchor the mechanism claims. Within-paper findings explain applicability in the evaluated settings, but do not establish a common task-performance ranking or the frequency with which a route succeeds across deployments.

9.2 Open Challenges

Preparation costs under the client budget. Proxy construction, similarity profiling, prealignment, quantization, and cache preparation can have different working sets from steady adaptation. A method that fits during local updates may require a larger graph earlier. A phase-specific account of resident weights, training states, activations, buffers, and their locations would expose this earlier constraint and show how preparation cost is amortized over reuse.

Comparable dynamic memory and concurrency. Shared server weights reduce duplication but leave independent optimizer states and queued features. Relating active graphs, resident sessions, and offloaded state to one memory timeline would make concurrency limits comparable across sequential, batched, and asynchronous implementations.

Feature refresh and adaptation capacity. Cached representations reduce recurrent interaction while

fixing part of the learning graph. Changes to client backbones, input distributions, or objectives can require refreshed features. For intermittent clients, a refresh policy must weigh the adaptation value of new representations against extraction, buffering, transmission, and retraining costs.

Attributing composition benefits. A complete combined protocol can improve a measured outcome without showing that each ingredient contributes independently. Matched configurations and component ablations should separate reduced object counts from costs transferred to auxiliary models, alignment, or servers. This is especially relevant when multiple components remove overlapping training states.

Quantifying the privacy–resource trade-off. Protection mechanisms are counted here as additions to the client, network, and server budgets, but the review does not measure how much protection a given disclosed object requires. Relating a stated threat model to the secure-aggregation, encryption, or noise cost needed for each disclosed object—adapter update, boundary activation, logit, or cached embedding—would let a configuration be screened for trustworthiness and resource feasibility together rather than in sequence, and would expose cases where the cheapest route to adapt is not the cheapest route to adapt *safely*.

9.3 Conclusions

A fully resident pretrained base sets a client-memory floor that reducing adapter rank cannot remove. Quantization, graph reduction, offloading, and partitioning alter this floor by changing representation or placement. Communication and server capacity then depend on workload and residency: boundary tensors grow with sequence length and interaction count, while shared weights leave private training states and caches. Combinations bring these constraints together and must be counted as complete protocols, from preparation through deployment. Because these routes are adopted for data protection, the same account should record what each one discloses: data locality constrains the raw dataset, but the exchanged object determines the residual privacy surface and the protection cost a trustworthy deployment must absorb within the same budgets. The resulting account connects the four routes to concrete client, network, server, and disclosure budgets; task-specific evidence, together with the applicable threat and reliability model, determines which feasible configuration offers a suitable and trustworthy form of adaptation.

Data Availability Statement

Not applicable.

Funding

This work was supported in part by the National Natural Science Foundation of China under Grant 62372110, and in part by the Fujian Provincial Natural Science Foundation under Grant 2023J02008.

Conflicts of Interest

Yunheng Shen is affiliated with JD.com, Inc., Beijing 101111, China. The authors declare that this affiliation had no influence on the study design, data collection, analysis, interpretation of the results, or the decision to publish. Yang Yang served as an Associate Editor of the *Journal of Reliable and Secure Computing* at the time of manuscript submission. To ensure the integrity and impartiality of the peer-review process, Yang Yang was not involved in the editorial handling, peer review, or decision-making for this manuscript, which was handled independently by another editor. The authors declare no other conflicts of interest.

AI Use Statement

The authors declare that ChatGPT-5.6 Sol was used for drafting and language editing of the manuscript. The authors have carefully reviewed, revised, and verified the AI-assisted output and take full responsibility for the content of the manuscript.

Ethical Approval and Consent to Participate

Not applicable.

References

[1] Woisetschläger, H., Isenko, A., Wang, S., Mayer, R., & Jacobsen, H. A. (2024). A survey on efficient federated learning methods for foundation model training. *arXiv preprint arXiv:2401.04472*. [CrossRef]

[2] Yan, N., Su, Y., Deng, Y., & Schober, R. (2025). Federated fine-tuning of LLMs: Framework comparison and research directions. *IEEE Communications Magazine, 63*(10), 52–58. [CrossRef]

[3] Wu, Y., Tian, C., Li, J., Sun, H., Tam, K., Zhou, Z., ... & Xu, C. (2025). A survey on federated fine-tuning of large language models. *arXiv preprint arXiv:2503.12016*. [CrossRef]

[4] Guo, T., Wang, J., Huo, F., Cui, L., Guo, S., Gui, J., & Tao, D. (2025). On the Evolution of Federated Post-Training Large Language Models: A Model Accessibility View. *arXiv preprint arXiv:2508.16261*. [CrossRef]

[5] Yang, Y., Long, G., Lu, Q., Zhu, L., Jiang, J., & Zhang, C. (2025). Federated low-rank adaptation for foundation models: A survey. *Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI)*, 10779–10787. [CrossRef]

[6] Bian, J., Peng, Y., Wang, L., Huang, Y., & Xu, J. (2025). A survey on parameter-efficient fine-tuning for foundation models in federated learning. *arXiv preprint arXiv:2504.21099*. [CrossRef]

[7] Yao, Y., Zhang, J., Wu, J., Huang, C., Xia, Y., Yu, T., ... & Joe-Wong, C. (2026). Federated large language models: Current progress and future directions. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (pp. 203-221). Springer, Singapore. [CrossRef]

[8] Liu, Z., Wang, Y., Wang, R., Tang, X., & Wu, S. (2026). A Survey on Split Learning for LLM Fine-Tuning: Models, Systems, and Privacy Optimizations. *arXiv preprint arXiv:2604.24468*. [CrossRef]

[9] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*. [CrossRef]

[10] Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems, 36*, 10088-10115. [CrossRef]

[11] Cho, Y. J., Liu, L., Xu, Z., Fahrezi, A., & Joshi, G. (2024). Heterogeneous LoRA for federated fine-tuning of on-device foundation models. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 12903–12913. [CrossRef]

[12] Wang, Z., Shen, Z., He, Y., Sun, G., Wang, H., Lyu, L., & Li, A. (2024). Flora: Federated fine-tuning large language models with heterogeneous low-rank adaptations. *Advances in Neural Information Processing Systems, 37*, 22513-22533. [CrossRef]

[13] Bai, J., Chen, D., Qian, B., Yao, L., & Li, Y. (2024). Federated fine-tuning of large language models under heterogeneous tasks and client resources. *Advances in Neural Information Processing Systems, 37*, 14457-14483. [CrossRef]

[14] Li, S., & Voigt, T. (2026). FraQ: Efficient Coordinate-Space Recompression for Federated Low-Rank Adaptation. *arXiv preprint arXiv:2608.03605*. [CrossRef]

[15] Sun, Y., Li, Z., Li, Y., & Ding, B. (2024, May). Improving lora in privacy-preserving federated learning. In *International conference on learning representations* (Vol. 2024, pp. 17978-17994).

[16] Babakniya, S., Elkordy, A. R., Ezzeldin, Y. H., Liu, Q., Song, K. B., El-Khamy, M., & Avestimehr, S. (2023). Slora: Federated parameter efficient fine-tuning of language models. *arXiv preprint arXiv:2308.06522*. [CrossRef]

[17] Zhang, Z., Hu, R., Liu, P., & Xu, J. (2024). Fed-pilot: Optimizing lora allocation for efficient federated fine-tuning with heterogeneous clients. *arXiv preprint arXiv:2410.10200*. [CrossRef]

[18] Su, Y., Yan, N., Deng, Y., Dohler, M., & Schober, R. (2026). HAFLQ: Heterogeneous Adaptive Federated LoRA Fine-tuned LLM with Quantization. *IEEE Transactions on Wireless Communications, 25*, 20054-20070. [CrossRef]

[19] JianHao, Z., Lv, C., Wang, X., Wu, M., Liu, W., Li, T., ... & Huang, X. J. (2024, November). Promoting data and model privacy in federated learning through quantized lora. In *Findings of the Association for Computational Linguistics: EMNLP 2024* (pp. 10501-10512). [CrossRef]

[20] Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J. D., Chen, D., & Arora, S. (2023). Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems, 36*, 53038-53075. [CrossRef]

[21] Xu, M., Cai, D., Wu, Y., Li, X., & Wang, S. (2024). {FwdLLM}: Efficient federated finetuning of large language models with perturbed inferences. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)* (pp. 579-596). https://www.usenix.org/conference/atc24/presentation/xu-mengwei

[22] Qin, Z., Chen, D., Qian, B., Ding, B., Li, Y., & Deng, S. (2023). Federated full-parameter tuning of billion-sized language models with communication cost under 18 kilobytes. *arXiv preprint arXiv:2312.06353*. [CrossRef]

[23] Li, Z., Ying, B., Liu, Z., Dong, C., & Yang, H. (2025). Achieving dimension-free communication in federated learning via zeroth-order optimization. *Proceedings of the International Conference on Learning Representations (ICLR)*, 23769-23803.

[24] Ling, Z., Chen, D., Yao, L., Li, Y., & Shen, Y. (2024, August). On the convergence of zeroth-order federated tuning for large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (pp. 1827-1838). [CrossRef]

[25] Ahmed, M. A., Pfeiffer, K., Khalili, R., Khdr, H., & Henkel, J. (2025). Efficient Zeroth-Order Federated Finetuning of Language Models on Resource-Constrained Devices. *arXiv preprint arXiv:2502.10239*. [CrossRef]

[26] Sun, J., Xu, Z., Yin, H., Yang, D., Xu, D., Chen, Y., & Roth, H. R. (2023). Fedbpt: Efficient federated black-box prompt tuning for large language models. *arXiv preprint arXiv:2310.01467*. [CrossRef]

[27] Yang, G., Wu, T., Guo, Y., Sun, Y., & Gong, Y. (2026, May). FedKRSO: Communication and memory

efficient federated fine-tuning of large language models. In *IEEE INFOCOM 2026-IEEE Conference on Computer Communications* (pp. 1-10). IEEE. [CrossRef]

[28] Shu, Y., Hu, W., Ng, S. K., Low, B. K. H., & Yu, F. R. (2024). Ferret: Federated full-parameter tuning at scale for large language models. *arXiv preprint arXiv:2409.06277*. [CrossRef]

[29] Xiao, G., Lin, J., & Han, S. (2023). Offsite-tuning: Transfer learning without full model. *arXiv preprint arXiv:2302.04870*. [CrossRef]

[30] Wu, F., Li, Z., Li, Y., Ding, B., & Gao, J. (2024, August). Fedbiot: Llm local fine-tuning in federated learning without full model. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining* (pp. 3345-3355). [CrossRef]

[31] Peng, Z., Fan, X., Chen, Y., Wang, Z., Pan, S., Wen, C., Zhang, R., & Wang, C. (2024). FedPFT: Federated proxy fine-tuning of foundation models. *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI)*, 4806–4814. [CrossRef]

[32] Wu, Y., Li, J., Tian, C., Guo, Z., & Li, L. (2025). Memory-efficient federated fine-tuning of large language models via layer pruning. *arXiv preprint arXiv:2508.17209*. [CrossRef]

[33] Zhu, M., Guo, S., Zhou, P., Ning, Y., Han, C., & Qiao, D. (2025). FedSODA: Federated fine-tuning of LLMs via similarity group pruning and orchestrated distillation alignment. *ECAI 2025, Frontiers in Artificial Intelligence and Applications, 413*. IOS Press. [CrossRef]

[34] Fan, T., Ma, G., Song, Y., Fan, L., Chen, K., & Yang, Q. (2026, July). FedProxy: Federated Fine-Tuning of LLMs via Proxy SLMs and Heterogeneity-Aware Fusion. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 17482-17497). [CrossRef]

[35] Gao, Z., Zhang, Y., Zhang, Z., Gong, Y., & Guo, Y. (2024). FedPT: federated proxy-tuning of large language models on resource-constrained edge devices. *arXiv preprint arXiv:2410.00362*. [CrossRef]

[36] Fan, T., Ma, G., Kang, Y., Gu, H., Song, Y., Fan, L., ... & Yang, Q. (2025, January). Fedmkt: Federated mutual knowledge transfer for large and small language models. In *Proceedings of the 31st International Conference on Computational Linguistics* (pp. 243-255). https://aclanthology.org/2025.coling-main.17/

[37] Lin, Z., Hu, X., Zhang, Y., Chen, Z., Fang, Z., Chen, X., ... & Gao, Y. (2024). Splitlora: A split parameter-efficient fine-tuning framework for large language models. *arXiv preprint arXiv:2407.00952*. [CrossRef]

[38] Lin, Z., Zhang, Y., Chen, Z., Fang, Z., Chen, X., Vepakomma, P., Ni, W., Luo, J., & Gao, Y. (2026). HSplitLoRA: A heterogeneous split parameter-efficient fine-tuning framework for large language models. *IEEE Transactions on Mobile Computing, 25*(10), 15543-15559. [CrossRef]

[39] Chen, X., Li, L., Ji, F., & Wu, W. (2025, May). Memory-efficient split federated learning for llm fine-tuning on heterogeneous mobile devices. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (pp. 1-6). IEEE. [CrossRef]

[40] Zhao, K., Yang, Z., Hu, Y., Chen, M., Zhu, C., & Zhang, Z. (2025). Efficient Split Federated Learning for Large Language Models over Communication Networks. *arXiv preprint arXiv:2504.14667*. [CrossRef]

[41] Shan, Y., Zhang, Z., Di, S., Liu, Y., Lu, X., & Liu, B. (2026). SplitFT: An Adaptive Federated Split Learning System For LLMs Fine-Tuning. *arXiv preprint arXiv:2604.26388*. [CrossRef]

[42] Geng, J., Yuan, T., Han, P., Gao, Y., Chen, X., & Luo, B. (2025). FlexP-SFT: A flexible aggregation-free framework for on-device personalized split federated fine-tuning of LLMs. *arXiv preprint arXiv:2508.10349*. [CrossRef]

[43] Kou, Z., Chen, Z., Yang, J., & Shen, C. (2026). Lean Clients, Full Accuracy: Hybrid Zeroth-and First-Order Split Federated Learning. *arXiv preprint arXiv:2601.09076*. [CrossRef]

[44] Geng, J., Chen, X., & Luo, B. (2026). Just Talk Once: Communication-Efficient Split Federated LLM Fine-Tuning on Edge Devices. *arXiv preprint arXiv:2609.01457*. [CrossRef]

[45] Yang, X., Li, L., Li, S., Guan, L., Wang, H., Qi, X., ... & Pan, M. (2025). Fed MobiLLM: Efficient Federated LLM Fine-Tuning over Heterogeneous Mobile Devices via Server Assisted Side-Tuning. *arXiv preprint arXiv:2508.06765*. [CrossRef]

Yunheng Shen received his B.Eng. and Ph.D. degrees from the Department of Automation, Tsinghua University, China, in 2019 and 2026, respectively. His current research interests include federated learning, diffusion models, and large language model inference.

Xiao Liu received his B.Eng. degree from the School of Information and Software Engineering, University of Electronic Science and Technology of China (UESTC), China. He is currently pursuing the M.Eng. degree at UESTC. His research interests include computer vision, visual perception, and dataset analysis.

Yang Yang received her Ph.D. degree from Xidian University, China, in 2011. She is a Senior Research Scientist with the School of Computing and Information Systems, Singapore Management University. She has also been a Full Professor with the College of Computer and Data Science, Fuzhou University. Her research interests include federated learning and privacy protection. She has published more than 100 papers in venues including NDSS, USENIX Security, TIFS, TDSC, TSC, TCC, and TII. She is an IEEE Senior Member.

Hairong Lv received his B.Eng. and Ph.D. degrees from Tsinghua University, Beijing, China, in 2002 and 2007, respectively. He worked as a researcher and manager at IBM Research, Beijing, and has also served as chairperson of technology companies. He has been an Associate Researcher with the Department of Automation, Tsinghua University, since 2017. His research interests include federated learning, pretrained large language models, knowledge graphs, and their applications to medicine and geoscience.