Analysis of Serverless Computing Implementation: Performance and Cost Optimization in Modern Architectures.\* (use style: *paper title*)

1st Raymond Senico   
Fakultas Ilmu Komputer  
Universitas Pelita Harapan  
Medan, Indonesia  
[03082220001@student.uph.edu](mailto:03082220001@student.uph.edu)

2nd Vania Callysta Tandy  
Fakultas Ilmu Komputer  
Universitas Pelita Harapan  
Medan, Indonesia  
[03082220007@student.uph.edu](mailto:03082220007@student.uph.edu)

3rd Stewart Fernandez  
Fakultas Ilmu Komputer  
Universitas Pelita Harapan  
Medan, Indonesia  
[03082220010@student.uph.edu](mailto:03082220010@student.uph.edu)

Abstract—Serverless computing is an innovative model in cloud architecture that simplifies application development with reduced costs. This study analyzes the implementation of serverless computing, focusing on performance and cost-saving strategies. We evaluate the system's operation, resource utilization, and pricing models to understand the relationship between architectural decisions and operational efficiency. Findings highlight key factors impacting performance and costs, such as function size, execution duration, and resource allocation in serverless computing applications

Keywords—Serverless computing, cloud architecture, cost optimization, pricing model, operational efficiency.

# Introduction (*Heading 1*)

Serverless computing has emerged as a cloud architecture model that transforms how organizations deploy and manage applications, allowing them to focus on code execution without handling server infrastructure. Although serverless computing offers significant advantages in cost and operational efficiency, optimizing performance and resource utilization in its implementation requires careful attention.

Key factors include (1) cost reduction potential through efficient resource allocation and pay-per-execution models, (2) automatic scaling capabilities that manage varying workloads, and (3) reduced operational burden by eliminating infrastructure management responsibilities. However, achieving optimal performance while maintaining cost efficiency demands careful consideration in implementation strategy.

Modern serverless platforms offer various approaches to resource management and pricing. Understanding the impact of factors such as cold starts, function execution patterns, and resource allocation strategies is crucial for successful implementation. This study addresses these challenges by examining the relationship between implementation choices and their effects on performance and cost metrics.

Through analyzing different implementation patterns and their implications, this research aims to provide practical insights for organizations to optimize serverless computing applications, aiding them in effectively leveraging this technology within modern cloud architectures.

# Research Methodology

## Serverless Store

Serverless Store is a web-based e-commerce demo application developed by Google to showcase serverless solutions on the Google Cloud platform. It allows users to list, browse, and purchase items and is built entirely with serverless architecture, using Google Cloud services like Google App Engine, Google Cloud Functions, and Google Cloud Pub/Sub. This architecture removes the need for server management, scales automatically based on demand, and incurs costs only when there is user activity

The app employs an event-driven design where workflows are triggered by events, such as user actions (e.g., placing orders) and system notifications relayed through Cloud Pub/Sub. This setup facilitates automated workflows, consistent logging, and seamless log storage. Although not an official Google product, this example demonstrates the flexibility and efficiency of serverless architecture in supporting high scalability and reliability on Google Cloud.

Serverless Store Link: <https://github.com/GoogleCloudPlatform/serverless-store-demo>

## Advantage of Serverless Computing

**Cost Efficiency**

A primary advantage of serverless computing is cost efficiency. The pay-as-you-go model ensures businesses are only charged based on actual resource usage rather than allocated server capacity. This approach eliminates idle server costs, enabling companies to optimize expenses in real time.

**Scalability**

Cloud providers automatically handle scaling based on application demand, ensuring smooth operations without manual intervention. This feature is beneficial for applications with fluctuating traffic, such as e-commerce sites during sales events or rapidly growing startups, as it meets dynamic demand without incurring the expenses and challenges of traditional scaling

**Accelerated Development Speed**

Serverless architecture speeds up software development. Without infrastructure management, developers can focus on code, dramatically reducing development cycles. Immediate deployment environments offered by serverless platforms facilitate rapid iteration, allowing businesses to respond swiftly to market changes, user feedback, and emerging trends.

**Reduced Operational Overhead**

Traditional computing models involve operational demands, from server management to software updates, diverting resources from core business goals. Serverless computing shifts these responsibilities to cloud providers, simplifying application deployment and eliminating complex server configurations. Additionally, rollback to previous application versions in case of issues becomes easier, enhancing business continuity.

## Disadvantages of Serverless Computing

**Cold Starts**   
A common limitation in serverless computing is the "cold start" phenomenon, where a new function instance incurs latency before execution, especially when the environment needs to be initialized. This can affect performance in applications requiring quick response times, potentially impacting user experience and engagement

**Vendor Lock-In**  
Adopting serverless architecture often requires adaptation to specific tools, infrastructure, and services from a particular provider. This dependency can result in vendor lock-in, where migrating to another platform becomes time-consuming and costly due to incompatible interfaces and configurations, limiting flexibility.

**Customization and Control Limitations**  
Serverless platforms often impose restrictions on runtime environment, memory limits, and execution duration. While simplifying many operations, serverless frameworks may not meet highly specialized use cases, requiring alternative solutions or workarounds

**Security Concerns**  
In serverless computing, while infrastructure security is managed by the provider, application security remains a shared responsibility. Serverless environments can introduce unique vulnerabilities, such as event-driven data injection or unauthorized function calls. These functions, especially those interacting with other cloud services, can become potential entry points, necessitating strict security practices in serverless architecture.

## Performance Evaluation

To assess serverless computing performance, this study conducts testing on various metrics, including execution latency, cold start frequency, and resource utilization under different workloads. This analysis helps in understanding how these factors influence response speed and application stability. Tests may include repeated usage scenarios over short periods versus sporadic use with longer intervals, offering insights into cold start impacts and resource allocation efficiency.

Performance evaluation methods may include:

* **Latency Measurement**: Measuring the average time taken for a serverless function to start and respond.
* **Cold Start Frequency**: Identifying the conditions and frequency of cold starts and ways to minimize them.
* **Memory and CPU Monitoring**: Observing resource utilization per execution to optimize performance and detect potential bottlenecks.

## Cost Analysis

Cost analysis will evaluate the effectiveness of the serverless pay-as-you-go model, including costs based on demand and the implications of function execution frequency and duration. To maximize cost efficiency, this research examines how appropriate memory settings and optimal function durations can lower operational costs without compromising performance.

Key factors analyzed include:

* **Effectiveness of Usage-Based Payment Models**: Understanding whether this model is beneficial in various workload scenarios.
* **Impact of Cold Starts on Costs**: Assessing if cold starts significantly add to costs at scale.
* **Resource Allocation Optimization**: Identifying configuration parameters (like memory allocation) that balance optimal performance with cost efficiency.

## Scalability testing

Scalability testing will observe how serverless architecture handles demand surges efficiently and automatically. This testing will also measure the effectiveness of automatic scaling in meeting dynamic traffic needs and identify potential challenges during scaling, particularly for time-sensitive applications.

Testing steps may include:

* **Simulating Demand Surges**: Measuring scaling capability during user traffic spikes.
* **Analyzing Scaling Impact on Latency**: Evaluating how scaling processes affect response time.
* **Evaluating Performance under High and Low Loads**: Identifying application performance limits in varying workloads.

# Implementation

The Serverless Store is an e-commerce demo application developed by Google that utilizes serverless architecture principles on the Google Cloud Platform (GCP). The application showcases the capabilities and efficiencies of serverless computing by providing a fully managed, auto-scaling environment where users can perform typical e-commerce activities such as listing, browsing, and purchasing items. Through the Serverless Store, Google illustrates how serverless architecture can reduce operational overhead, enhance scalability, and offer cost-effective solutions for cloud-based applications.

## Fully Serverless Architecture

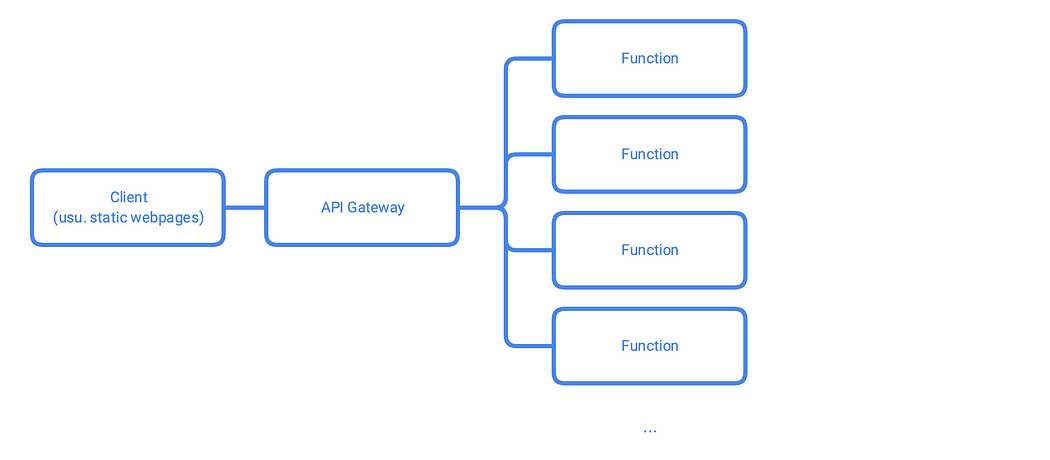
The Serverless Store is architected entirely with Google-managed serverless services, including Google App Engine, Google Cloud Functions, Google Cloud Pub/Sub, and other GCP tools. This design eliminates the need for any traditional server management, allowing developers to focus solely on application logic rather than infrastructure. Each of these services contributes to a seamless, scalable environment:

* **Google App Engine** handles web requests and offers a platform-as-a-service (PaaS) solution for building and deploying applications. It scales automatically based on traffic and user demand, allowing the Serverless Store to accommodate varying levels of activity without manual intervention.
* **Google Cloud Functions** enables the execution of code in response to specific events. This allows developers to create modular, event-driven functions that respond to user actions or system events without provisioning servers. For instance, when a user places an order, a Cloud Function may be triggered to process the order, ensuring that functions are executed only when necessary.
* **Google Cloud Pub/Sub** serves as the messaging system that relays notifications and events across the application. It enables real-time communication between different components, facilitating a highly responsive, event-driven architecture that drives workflows within the Serverless Store.

This combination of serverless services not only automates scalability but also enforces a pay-per-use pricing model, where costs are incurred only when resources are actively used. This model is a key advantage of serverless computing, as it significantly reduces expenses, especially for applications with fluctuating workloads.

## Serverless and Faas

The implementation of serverless computing architecture in Serverless Store adopts the Function as a Service (FaaS) paradigm as a fundamental component in system development. This approach enables more efficient application development by breaking down application functions into smaller, independent units. Each function is designed to handle specific tasks, such as product management, order processing, or inventory management, which can be deployed and managed separately.



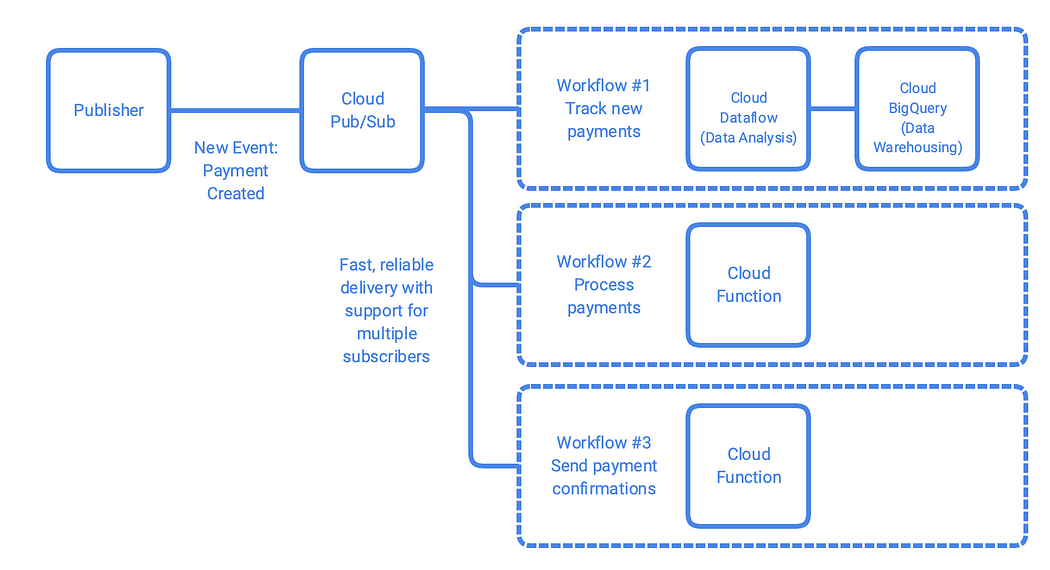
In its implementation, the system architecture is built utilizing an API Gateway as the main entry point that manages and directs requests to appropriate functions. The API Gateway not only functions as a router but also provides a centralized security and access management layer. The user interface is implemented using static web pages distributed through a Content Delivery Network (CDN), ensuring optimal performance and fast response times.

The main advantage of this implementation is the system's ability to automatically scale based on workload. When there is an increase in demand, the serverless platform automatically adds capacity to handle the load, and conversely will reduce capacity when demand decreases. This results in cost efficiency as payment is only made based on actual resource usage.

## Event-Driven Architecture

The implementation of event-driven architecture in Serverless Store is a key aspect that enables the system to handle operations asynchronously and in real-time. This architecture is built on top of a publish/subscribe system that enables effective communication between system components. When an event occurs, such as creating a new order, the system will publish this event to a message queue, which will then distribute the information to various subscribers that need it.

In its application, each system component acts as an independent publisher and/or subscriber. For example, when an order is created, the Order Service will publish an ORDER\_CREATED event. This event will then be received by various subscribers such as Inventory Service to update stock, Notification Service to send confirmations to customers, and Analytics Service to record transaction data. This approach allows the system to process various tasks in parallel and maintain optimal performance.



One important aspect of event-driven implementation is the ability to perform real-time analytics. By utilizing continuous event streams, the system can process and analyze data in real-time, providing valuable insights about system performance and user behavior. This enables development and business teams to make decisions based on current data and respond quickly to changing needs

## Cost-Effective Storage Options

Serverless Store employs Cloud Firestore and Cloud Datastore for storage, which scale dynamically based on demand. These systems are chosen for their efficiency in managing structured and unstructured data, particularly for applications with fluctuating workloads. To optimize costs, regional settings can be leveraged; for example, using multi-regional configurations for high availability and regional configurations for cost savings. Lifecycle rules and automated backups are implemented to manage storage effectively by archiving unused data, reducing expenses. Proper index configurations prevent unnecessary querying costs, aligning with the pay-per-use pricing model that minimizes idle resource charges.

## Commited Use Discount

Google Cloud provides Committed Use Discounts (CUDs), enabling significant savings (up to 57%) when users commit to specific resource levels over 1- or 3-year terms. For most project, storage and BigQuery workloads benefit the most. An analysis of workloads determined that predictable tasks like real-time analytics using BigQuery fit well within CUD frameworks.

## FinOps Integration

Financial operations (FinOps) emphasize collaboration between finance and engineering teams to balance cost efficiency and performance. Using tagging and account-level segmentation helps in allocating and optimizing spending, Leveraging Google Cloud Billing Reports with FinOps allows setting alerts for overspending and detailed insight into expensive operations like data export.

## Regional Pricing

Pricing for serverless services varied significantly across regions, impacting overall cost planning. Deployment in low-cost regions such as certain Asian or South American locations, can reduce expenses by 30% but may increase latency depending on user proximity.

## BigQuery Cost Optimization

BigQuery is continuously used in Google Cloud Platform Project for analytics, to optimize the usage there are several tips :

* Partitioned Tables: Use time-based partitioning for logs and transaction data, ensuring queries run on smaller datasets.
* Query Pricing Model: Shift to flat-rate pricing for high query volumes to cap costs while ensuring scalability.
* Materialized Views: Precompute frequent queries, reducing processing overhead.

## Vertex Ai for Cost and Performance Optimization

Vertex AI is Google Cloud's comprehensive platform for building, deploying, and managing machine learning (ML) models. Unlike traditional approaches where ML workflows are fragmented across multiple tools, Vertex AI unifies the entire ML lifecycle within a single environment. It integrates data preparation, model training, hyperparameter tuning, deployment, monitoring, and governance. This platform supports both pre-trained models and custom models, making it versatile for developers and data scientists alike. Its design emphasizes scalability, efficiency, and ease of use, enabling organizations to develop robust ML solutions while minimizing operational overhead.

Vertex AI is used to streamline the development and deployment of machine learning models by offering a variety of tools and features:

1. Data Preparation and AutoML: Vertex AI automates data preprocessing tasks and allows developers to create models using AutoML, a feature that automatically selects and optimizes the best algorithms for a given dataset. This significantly reduces the time and expertise needed for model training.
2. Custom Model Training: Developers can build and train their custom models .Vertex AI provides managed resources, such as GPUs and TPUs, to optimize computational efficiency.
3. Model Deployment and Serving: The platform offers serverless model deployment, which scales resources automatically based on traffic demands. This ensures low-latency predictions without requiring constant infrastructure management.
4. Vertex Pipelines: Automates end-to-end workflows, including data ingestion, training, and validation. This eliminates manual steps and ensures consistent execution across iterations.
5. Explainable AI: Vertex AI provides tools to interpret model predictions, improving transparency and trust, which is crucial for decision-making in industries like healthcare and finance.
6. Monitoring : Built-in features for monitoring model performance over time help identify data drift and accuracy degradation, allowing developers to address issues proactively.

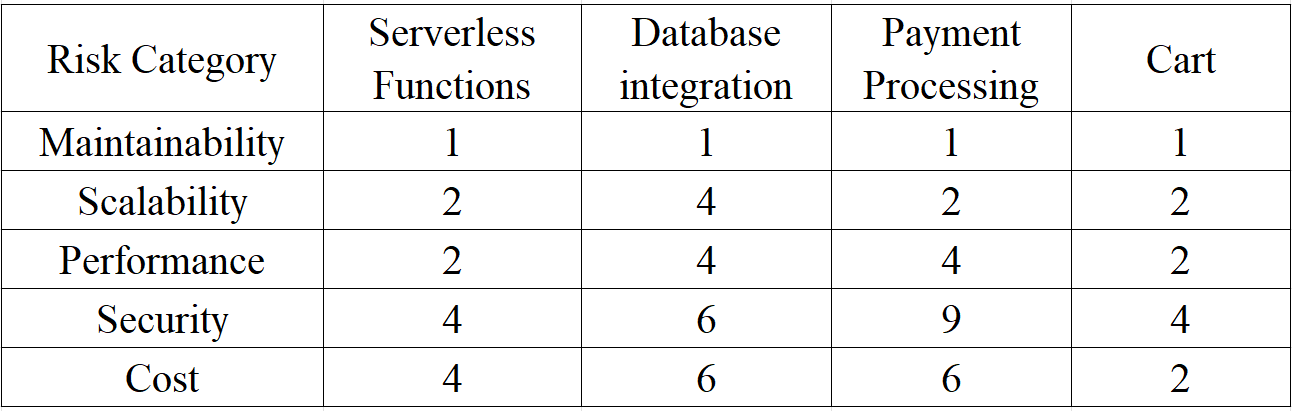
Vertex AI is specifically designed to enhance both performance and cost efficiency. Its serverless nature ensures that resources are used only when required, aligning closely with the "pay-per-use" model of cloud computing. For example, during model training, Vertex AI dynamically allocates compute resources based on workload intensity. This avoids over-provisioning, which is a common cost driver in traditional infrastructure setups.

The platform also supports optimized model training through hyperparameter tuning and distributed training, enabling faster results with fewer computational cycles. By deploying models closer to end-users in multi-region setups, latency is reduced, improving performance. Vertex Pipelines automate workflows, reducing time and human error, while Explainable AI prevents overfitting and ensures that models are efficient and interpretable.

## Monitoring and Cost Management

* Google Cloud's Billing Reports : Use the Google Cloud Console’s built-in billing reports to monitor and analyze your cloud costs and usage. Break down costs by service, project, region, and label.
* Budgets and Alerts : Set up budgets and alerts to monitor your spending and get notified when your usage exceeds a specific threshold.
* Google Cloud Cost Management Tools: Leverage tools like Google Cloud Cost Management (formerly Cloud Billing Reports and Cloud Cost Insights) for deep analysis of cost patterns.

## Risk Assesment



1. Maintainability

* Serverless Functions: Low risk (value: 1), as serverless functions are easy to maintain without managing infrastructure.
* Database Integration: Low risk (value: 1), provided database integration is well-configured.
* Payment Processing: Low risk (value: 1), though it requires regular monitoring to avoid errors.
* Cart: Low risk (value: 1), as cart features are typically simple to maintain with a straightforward design.

2. Scalability

* Serverless Functions: Medium risk (value: 2), auto-scaling capabilities exist but may have certain limitations.
* Database Integration: High risk (value: 4), as scalability depends heavily on database design and query load.
* Payment Processing: Medium risk (value: 2), requiring a design that handles increased transactions efficiently.
* Cart: Medium risk (value: 2), especially during traffic spikes with concurrent users.

3. Performance

* Serverless Functions: Medium risk (value: 2), performance can be affected by cold starts.
* Database Integration: High risk (value: 4), performance depends on query optimization and database structure.
* Payment Processing: High risk (value: 4), requiring low latency and high reliability.
* Cart: Medium risk (value: 2), as quick response times are crucial for user experience.

4. Security

* Serverless Functions: High risk (value: 4), requiring strict access controls and data encryption.
* Database Integration: High risk (value: 6), as sensitive data demands extra layers of protection.
* Payment Processing: Very high risk (value: 9), as it is a primary target for cyber threats.
* Cart: High risk (value: 4), while cart data is less sensitive, it still needs to be secured against exploitation.

5. Cost

* Serverless Functions: High risk (value: 4), as costs may rise with excessive usage.
* Database Integration: High risk (value: 6), with costs increasing based on data volume and query load.
* Payment Processing: High risk (value: 6), depending on transaction volumes and payment gateway fees.
* Cart: Low risk (value: 2), as the cost of managing a cart is relatively low compared to other components.

Identify applicable funding agency here. If none, delete this text box.

* Use a zero before decimal points: “0.25”, not “.25”. Use “cm3”, not “cc”. (*bullet list*)
* The word “data” is plural, not singular.
* The subscript for the permeability of vacuum **0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter “o”.
* In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
* A graph within a graph is an “inset”, not an “insert”. The word alternatively is preferred to the word “alternately” (unless you really mean something that alternates).
* Do not use the word “essentially” to mean “approximately” or “effectively”.
* In your paper title, if the words “that uses” can accurately replace the word “using”, capitalize the “u”; if not, keep using lower-cased.
* Be aware of the different meanings of the homophones “affect” and “effect”, “complement” and “compliment”, “discreet” and “discrete”, “principal” and “principle”.
* Do not confuse “imply” and “infer”.
* The prefix “non” is not a word; it should be joined to the word it modifies, usually without a hyphen.
* There is no period after the “et” in the Latin abbreviation “et al.”.
* The abbreviation “i.e.” means “that is”, and the abbreviation “e.g.” means “for example”.

An excellent style manual for science writers is [7].

# Using the Template

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

## Authors and Affiliations

**The template is designed for, but not limited to, six authors.** A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

### For papers with more than six authors: Add author names horizontally, moving to a third row if needed for more than 8 authors.

### For papers with less than six authors: To change the default, adjust the template as follows.

#### Selection: Highlight all author and affiliation lines.

#### Change number of columns: Select the Columns icon from the MS Word Standard toolbar and then select the correct number of columns from the selection palette.

#### Deletion: Delete the author and affiliation lines for the extra authors.

## Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is “Heading 5”. Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract”, will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named “Heading 1”, “Heading 2”, “Heading 3”, and “Heading 4” are prescribed.

## Figures and Tables

#### Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1”, even at the beginning of a sentence.

1. Table Type Styles

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| copy | More table copya |  |  |

1. Sample of a Table footnote. (*Table footnote*)
2. Example of a figure caption. (*figure caption*)

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

##### Acknowledgment *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

##### References

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

1. *K. Mahajan, D. Figueiredo, V. Misra and D. Rubenstein, "Optimal Pricing for Serverless Computing," 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 2019, pp. 1-6, doi: 10.1109/GLOBECOM38437.2019.9013156. keywords: {Cloud computing;Load modeling;Computational modeling;Time factors;Containers;Pricing;Resource management},*
2. C. Lin and H. Khazaei, "Modeling and Optimization of Performance and Cost of Serverless Applications," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 3, pp. 615-632, 1 March 2021, doi: 10.1109/TPDS.2020.3028841. keywords: {FAA;Cloud computing;Optimization;Time factors;Analytical models;Computational modeling;Cloud serverless computing;performance modeling;performance optimization;cost modeling;cost optimization},
3. Thatikonda, V. K. (2023). Serverless Computing: Advantages, Limitations and Use Cases. *European Journal of Theoretical and Applied Sciences*, *1*(5), 341-347. <https://doi.org/10.59324/ejtas.2023.1(5).25>
4. Baldini, I. *et al.* (2017). Serverless Computing: Current Trends and Open Problems. In: Chaudhary, S., Somani, G., Buyya, R. (eds) Research Advances in Cloud Computing. Springer, Singapore. <https://doi.org/10.1007/978-981-10-5026-8_1>
5. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
6. K. Elissa, “Title of paper if known,” unpublished.
7. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
8. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
9. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

**IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.**

We suggest that you use a text box to insert a graphic (which is ideally a 300 dpi TIFF or EPS file, with all fonts embedded) because, in an MSW document, this method is somewhat more stable than directly inserting a picture.

To have non-visible rules on your frame, use the MSWord “Format” pull-down menu, select Text Box > Colors and Lines to choose No Fill and No Line.