Shopify Slideshow Requirements (with Code Reference & Visual Explanation)

Overview

This document details the exact requirements for the Shopify-integrated slideshow, ensuring a seamless and functional experience across desktop and mobile. The provided **images visually illustrate key issues and expected behavior**, while the **previously used code serves as a reference** for the level of depth and customization expected in the final implementation.

Core Functionality

✅ Desktop Behavior

- The slideshow **must display exactly 4 full images at a time on desktop** (not 3.5, not 4.2—precisely 4).
- **Each image should be fully visible**—no partial images should appear.
- **Autoplay transitions every 5 seconds**, ensuring smooth movement with no abrupt snapping.
- **70px spacing between images** to prevent them from touching.

✅ Mobile Behavior

- **Only 1 image should be displayed at a time on mobile**—no partial images should be visible.
- **No spacing on mobile**—each image should properly fill the screen width.
- **Autoplay should be continuous and always scroll to the right** (illustrated in **Image 2**).

✅ Looping & Scroll Behavior

- **Slideshow must loop infinitely**—once it reaches the last image, it should seamlessly transition back to the first **without blank spaces** (illustrated in **Image 1**).
- **On mobile, it must always scroll to the right**.
- **On desktop, continuous scrolling only activates if 8 or more images are uploaded**:
    - Once the **8th image enters view, the next transition should seamlessly cycle back to the 1st image from the right** (illustrated in **Image 2**).
    - If **fewer than 8 images** are uploaded, the scroll should **return from the left** (illustrated in **Image 3**).
    - **Regardless of the number of images, mobile should always maintain rightward scrolling.**

✅ No Empty Spaces

- **Image 1** highlights a critical issue where blank spaces appear instead of the slideshow seamlessly continuing—this must be prevented.

✅ Exact Image Dimensions

- The slideshow images must be **1311 x 1952px**, as this is the size being used for the brand's visuals.

Navigation & Interaction

✅ User Control Features

- The slideshow must include **left and right pagination arrows** for manual navigation.
- Users must be able to **click and drag (on desktop) or swipe (on mobile) to scroll seamlessly**.

✅ Direction Control

- **Mobile scrolling must always go to the right** (illustrated in **Image 2**).
- **Desktop scrolling must also go to the right if 8 or more images exist** (illustrated in **Image 2**).
- **If fewer than 8 images exist on desktop, scrolling should return from the left** (illustrated in **Image 3**).

Margins & Layout

✅ Visual Balance

- The slideshow must have **equal top and bottom margins** for aesthetic consistency.
- **Left and right margins** should ensure that the images do not hug the screen edges.

Integration & Customization

✅ Shopify Integration

- This slideshow must be **fully integrated into Shopify** with no external dependencies.

✅ Customization & Ease of Editing

- The system should allow **easy editing via a no-code UI**—users should not need to modify raw code to adjust images, spacing, or autoplay settings.

✅ **Brand-New Code Implementation**

- The previous **pasted code** was used as a reference and provided a solid base for customization, but due to issues in achieving the required scroll direction and looping behavior, a **brand-new codebase must be developed**.
- The new code should maintain the same **level of depth, settings, and customizability as the previous one**, ensuring full control over autoplay speed, margins, slide transition modes, navigation arrows, and pagination dots.

Visual References

- **Image 1:** Shows the issue of images not repopulating properly, leading to blank spaces. This must be **fixed so that the slideshow loops seamlessly**.
- **Image 2:** Shows the desired **rightward scrolling direction for mobile and for desktop when 8+ images are uploaded**.
- **Image 3:** Shows the behavior when **fewer than 8 images are present—scrolling should return from the left instead of continuously looping from the right**.

This is the **most detailed breakdown of the slideshow functionality** to ensure perfect execution. The reference code serves as a **benchmark for customization**, and the images highlight the **exact behaviors to fix**. The new code should be built from the ground up to **eliminate issues with cut-off images, inconsistent spacing, and improper looping behavior** while maintaining **smooth, user-friendly interactions**.

This must be executed precisely—no blank spaces, no abrupt transitions, and seamless looping based on the number of images uploaded.

Images In Question:

Image 1:

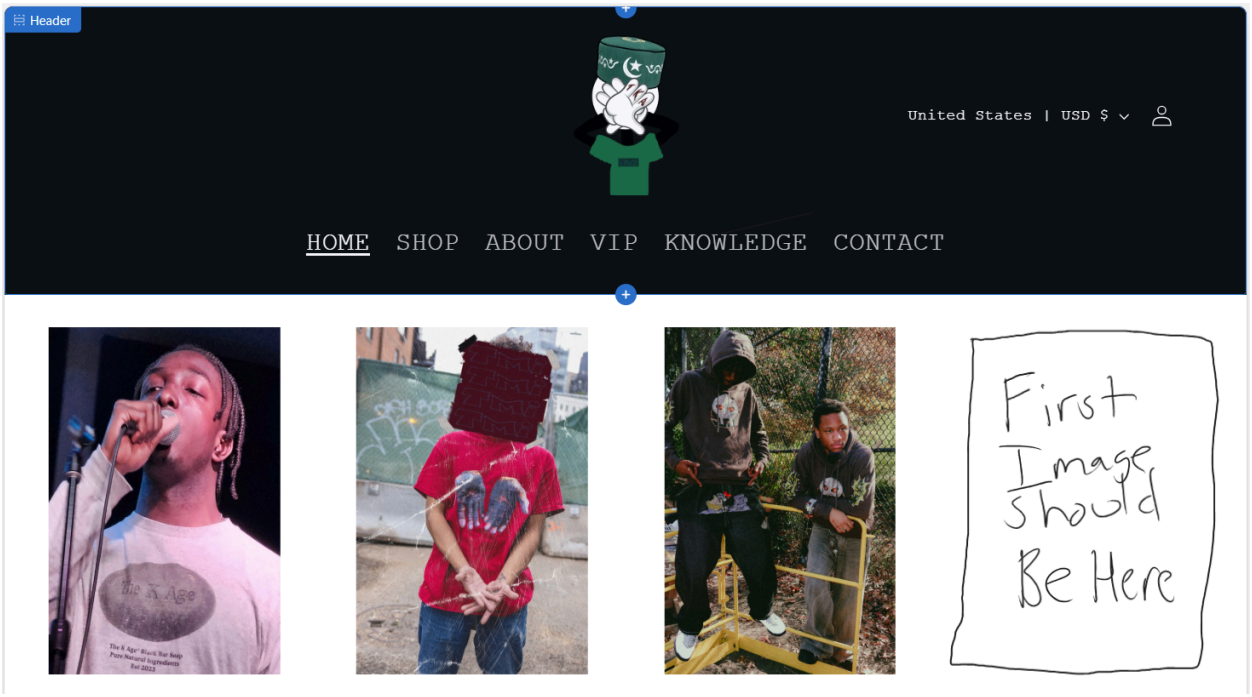


Image 2:

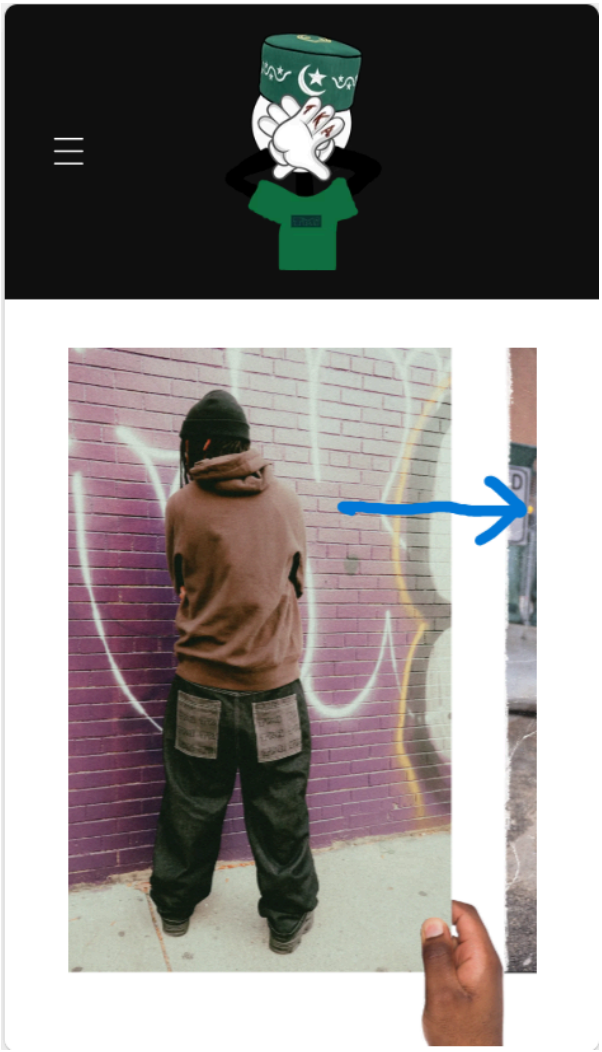
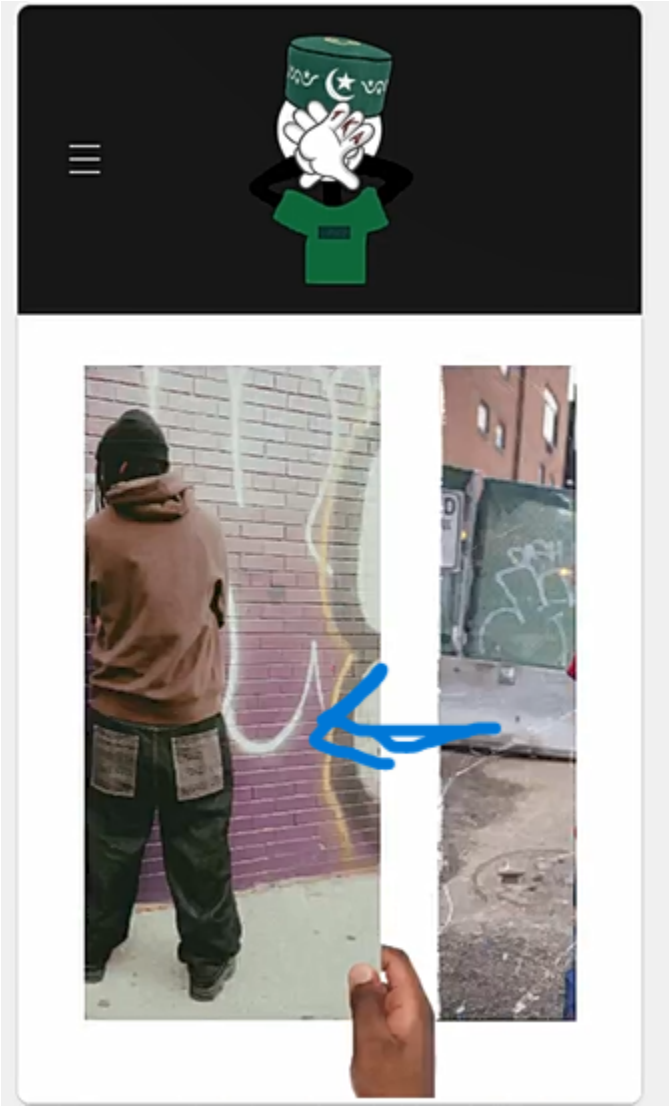


Image 3:



Original Code:

```
{% schema %}
{
  "name": "Custom Slideshow",
  "settings": [
    {
      "type": "range",
      "id": "autoplay_speed",
      "min": 3,
      "max": 9,
      "step": 1,
      "unit": "sec",
      "label": "Change slides every",
      "default": 5
    },
    {
      "type": "range",
      "id": "top_margin",
      "min": 10,
      "max": 100,
      "step": 5,
      "unit": "px",
      "label": "Top margin",
      "default": 30
    },
    {
      "type": "range",
      "id": "side_margin",
      "min": 10,
      "max": 100,
      "step": 5,
      "unit": "px",
      "label": "Side margins",
      "default": 40
    },
    {
      "type": "checkbox",
      "id": "show_arrows",
      "label": "Show arrows",
      "default": true
    },
```

```
  {
    "type": "checkbox",
    "id": "show_dots",
    "label": "Show dots",
    "default": true
  },
  {
    "type": "select",
    "id": "slide_mode",
    "label": "Slide transition mode",
    "options": [
      {
        "value": "slide_by_one",
        "label": "Slide by one image"
      },
      {
        "value": "slide_by_group",
        "label": "Slide by group"
      }
    ],
    "default": "slide_by_one"
  }
],
"blocks": [
  {
    "type": "slide",
    "name": "Slide",
    "settings": [
      {
        "type": "image_picker",
        "id": "image",
        "label": "Image"
      },
      {
        "type": "url",
        "id": "link",
        "label": "Link",
        "info": "Optional"
      },
      {
        "type": "select",
        "id": "alignment",
        "label": "Image alignment",
        "options": [
```

```
          {
            "value": "center",
            "label": "Center"
          },
          {
            "value": "top",
            "label": "Top"
          },
          {
            "value": "bottom",
            "label": "Bottom"
          },
          {
            "value": "left",
            "label": "Left"
          },
          {
            "value": "right",
            "label": "Right"
          }
        ],
        "default": "center"
      }
    ]
  }
],
"presets": [
  {
    "name": "Custom Slideshow",
    "category": "Image",
    "blocks": [
      {
        "type": "slide"
      },
      {
        "type": "slide"
      },
      {
        "type": "slide"
      },
      {
        "type": "slide"
      }
    ]
```

```
    }
  ]
}
{% endschema %}

<div
  class="custom-slideshow-container"
  style="padding-top: {{ section.settings.top_margin }}px; padding-left: {{
section.settings.side_margin }}px; padding-right: {{ section.settings.side_margin }}px;"
>
  <div id="custom-slideshow-wrapper" class="custom-slideshow-wrapper">
    <div id="custom-slideshow" class="custom-slideshow">
      {% for block in section.blocks %}
        {% if block.type == 'slide' and block.settings.image %}
          <div class="custom-slide" {{ block.shopify_attributes }}>
            {% if block.settings.link %}
              <a href="{{ block.settings.link }}">
            {% endif %}
            <div class="slide-image-wrapper" style="background-position: {{ block.settings.alignment
}};">
              <img
                src="{{ block.settings.image | img_url: 'master' }}"
                alt="{{ block.settings.image.alt | escape }}"
                class="slide-image"
              >
            </div>
            {% if block.settings.link %}
              </a>
            {% endif %}
          </div>
        {% endif %}
      {% endfor %}
    </div>
  </div>

  {% if section.settings.show_dots %}
    <div class="slide-dots"></div>
  {% endif %}

  {% if section.settings.show_arrows %}
    <div class="slide-arrow prev-arrow">&#10094;</div>
    <div class="slide-arrow next-arrow">&#10095;</div>
  {% endif %}
</div>
```

```
<style>
  .custom-slideshow-container {
    width: 100%;
    max-width: 100%;
    overflow: hidden;
    margin: 0 auto;
    position: relative;
  }

  .custom-slideshow-wrapper {
    width: 100%;
    overflow: hidden;
  }

  .custom-slideshow {
    display: flex;
    transition: transform 0.5s ease;
  }

  .custom-slide {
    flex-shrink: 0;
    box-sizing: border-box;
  }

  /* Desktop styles */
  @media screen and (min-width: 768px) {
    .custom-slide {
      width: calc(25% - 52.5px); /* Exactly 25% of width minus spacing */
      margin-right: 70px; /* Exactly 70px spacing as requested */
    }
  }

  /* Mobile styles */
  @media screen and (max-width: 767px) {
    .custom-slide {
      width: 100%;
      margin-right: 0;
    }
  }

  .slide-image-wrapper {
    width: 100%;
    /* Fixed aspect ratio based on 1311×1952 dimensions */
```

```
  padding-bottom: 148.89%; /* (1952/1311)*100 = 148.89% */
  position: relative;
  overflow: hidden;
}

.slide-image {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  object-fit: contain; /* Keep exact dimensions */
  border-radius: 0; /* No rounded edges */
}

/* Navigation controls */
.slide-arrow {
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
  cursor: pointer;
  z-index: 10;
  background: rgba(255, 255, 255, 0.7);
  width: 40px;
  height: 40px;
  display: flex;
  align-items: center;
  justify-content: center;
}

.prev-arrow {
  left: 10px;
}

.next-arrow {
  right: 10px;
}

.slide-dots {
  display: flex;
  justify-content: center;
  margin-top: 15px;
}
```

```
  .slide-dot {
    width: 10px;
    height: 10px;
    background: #ccc;
    border-radius: 50%;
    margin: 0 5px;
    cursor: pointer;
  }

  .slide-dot.active {
    background: #333;
  }

  /* Custom clone slide for continuous mobile sliding */
  .custom-slide.clone {
    display: none;
  }

  @media screen and (max-width: 767px) {
    .custom-slide.clone {
      display: block;
    }
  }
</style>

<script>
  document.addEventListener('DOMContentLoaded', function() {
    const slideshow = document.getElementById('custom-slideshow');
    const slidesArray = Array.from(document.querySelectorAll('.custom-slide'));
    const totalSlides = slidesArray.length;
    const autoplaySpeed = {{ section.settings.autoplay_speed | times: 1000 }};
    const slideByOne = {{ section.settings.slide_mode | json }} === "slide_by_one";

    if (totalSlides <= 0) return;

    let currentSlide = 0;
    let slidesPerView = 4; // Desktop default
    let slideWidth = 0;
    let slideMargin = 70; // As defined in CSS
    let cloneCreated = false;

    const isMobile = () => window.innerWidth < 768;

    // Function to create clone slides for mobile continuous scrolling
```

```
  function createMobileClones() {
    if (cloneCreated) return;

    // Only create clone on mobile
    if (isMobile()) {
      cloneCreated = true;

      // Create a clone of the first slide and append it to the end for continuous effect
      const firstSlideClone = slidesArray[0].cloneNode(true);
      firstSlideClone.classList.add('clone');
      slideshow.appendChild(firstSlideClone);
    }
  }

  // Function to calculate slide positions and widths
  function updateDimensions() {
    slidesPerView = isMobile() ? 1 : 4;

    if (isMobile()) {
      // Create a mobile clone if needed
      createMobileClones();

      slideWidth = slidesArray[0].offsetWidth;
      slideMargin = 0;
    } else {
      slideWidth = slidesArray[0].offsetWidth;
      slideMargin = 70; // Match the CSS value
    }

    // FIX: Ensure slideWidth is not zero
    if (slideWidth <= 0) {
      // Use container width instead if slide width is zero
      const containerWidth = document.querySelector('.custom-slideshow-wrapper').offsetWidth;
      slideWidth = isMobile() ? containerWidth : (containerWidth - (slidesPerView - 1) *
slideMargin) / slidesPerView;
    }

    // Ensure we're at a valid position after resize
    if (currentSlide > totalSlides - slidesPerView) {
      currentSlide = Math.max(0, totalSlides - slidesPerView);
      updateSlidePosition(false);
    } else {
      updateSlidePosition(false);
    }
```

```
    // Update dots
    createDots();
  }

// Replace your current updateSlidePosition function with this:
function updateSlidePosition(animate = true) {
  // Calculate how many slides are visible
  const visibleSlides = isMobile() ? 1 : slidesPerView;

  const translateX = currentSlide * (slideWidth + slideMargin);
  slideshow.style.transition = animate ? 'transform .5s ease' : 'none';
  slideshow.style.transform = `translateX(-${translateX}px)`;

  // Update dots
  updateDots();

  // Handle the special case for desktop continuous sliding
  if (!isMobile() && animate && currentSlide >= totalSlides - visibleSlides) {
    // Use a flag to prevent multiple event handling
    let resetInProgress = false;

    const handleTransitionEnd = function() {
      // Prevent multiple executions
      if (resetInProgress) return;
      resetInProgress = true;

      // Remove event listener first to prevent any chance of multiple calls
      slideshow.removeEventListener('transitionend', handleTransitionEnd);

      // Remove any existing clones
      document.querySelectorAll('.temp-clone').forEach(clone => clone.remove());

      // Reset to beginning without animation
      currentSlide = 0;
      slideshow.style.transition = 'none';
      slideshow.style.transform = 'translateX(0)';

      // Force reflow
      void slideshow.offsetHeight;

      // Wait a moment before re-enabling transitions
      setTimeout(() => {
        slideshow.style.transition = 'transform .5s ease';
```

```
      }, 50);
    };

    // Add the event listener for transition end
    slideshow.addEventListener('transitionend', handleTransitionEnd, {once: true});
  }

  // Mobile-specific reset logic
  if (isMobile() && animate && currentSlide === totalSlides) {
    setTimeout(() => {
      currentSlide = 0;
      slideshow.style.transition = 'none';
      slideshow.style.transform = 'translateX(0)';
      void slideshow.offsetHeight;
      slideshow.style.transition = 'transform .5s ease';
      updateDots();
    }, 500);
  }
}
  // For desktop continuous sliding: Handle the special reset case
  else if (!isMobile() && animate && currentSlide >= totalSlides - slidesPerView) {
    // Listen for the end of the transition
    const resetPositionHandler = function() {
      // Remove the temporary clones
      document.querySelectorAll('.temp-clone').forEach(clone => clone.remove());

      // Reset to beginning without animation
      currentSlide = 0;
      slideshow.style.transition = 'none';
      slideshow.style.transform = 'translateX(0)';

      // Force reflow
      slideshow.offsetHeight;

      // Restore transition for next slide
      slideshow.style.transition = 'transform .5s ease';

      // Update dots after resetting
      updateDots();

      // Restart autoplay
      startAutoplay();

      // Remove this event listener
```

```
      slideshow.removeEventListener('transitionend', resetPositionHandler);
    };

    slideshow.addEventListener('transitionend', resetPositionHandler, {once: true});
  }

  // Force reflow if not animating
  if (!animate) {
    slideshow.offsetHeight;
    slideshow.style.transition = 'transform .5s ease';
  }
}
// Function to go to a specific slide
function goToSlide(index, animate = true) {
  if (isMobile()) {
    // For mobile: ALWAYS continuous right sliding
    if (index >= totalSlides) {
      // Add a temporary clone to continue the rightward motion
      const firstClone = slidesArray[0].cloneNode(true);
      firstClone.classList.add('temp-clone');
      slideshow.appendChild(firstClone);
      // Let the animation to the clone complete
      currentSlide = index;
      updateSlidePosition(true);
      // After animation completes, instantly jump back to the first slide
      setTimeout(() => {
        // Remove the temporary clone
        slideshow.removeChild(firstClone);
        // Jump back to first slide without animation
        currentSlide = 0;
        updateSlidePosition(false);
      }, 500); // Match transition time
      return; // Exit early since we handled the animation separately
    } else if (index < 0) {
      // When going back from first slide on mobile
      currentSlide = totalSlides - 1;
    } else {
      currentSlide = index;
    }
  } else {
    // For desktop: Always use continuous sliding approach (simplified)
    if (index < 0) {
      currentSlide = totalSlides - slidesPerView;
    } else {
```

```
      currentSlide = index;
    }
  }
  updateSlidePosition(animate);
}
    // Function to go to next slide
    function nextSlide() {
      if (slideByOne) {
        // Slide by one image
        goToSlide(currentSlide + 1);
      } else {
        // Slide by group (slidesPerView)
        goToSlide(currentSlide + slidesPerView);
      }
    }

    // Function to go to previous slide
    function prevSlide() {
      if (slideByOne) {
        // Slide by one image
        goToSlide(currentSlide - 1);
      } else {
        // Slide by group (slidesPerView)
        goToSlide(currentSlide - slidesPerView);
      }
    }

    // Create dots based on visible slides
    function createDots() {
      if (!{{ section.settings.show_dots }}) return;

      const dotsContainer = document.querySelector('.slide-dots');
      dotsContainer.innerHTML = ''; // Clear existing dots

      // If sliding by one, create a dot for each slide
      // If sliding by group, create a dot for each complete group
      const totalDots = slideByOne ?
        totalSlides :
        Math.ceil(totalSlides / slidesPerView);

      for (let i = 0; i < totalDots; i++) {
        const dot = document.createElement('div');
        dot.className = 'slide-dot';
```

```
    const dotIndex = slideByOne ? i : i * slidesPerView;

    dot.addEventListener('click', function() {
      goToSlide(dotIndex);
    });

    dotsContainer.appendChild(dot);
  }

  updateDots();
}

// Update dots active state
function updateDots() {
  if (!{{ section.settings.show_dots }}) return;

  const dots = document.querySelectorAll('.slide-dot');

  dots.forEach((dot, i) => {
    if (slideByOne) {
      // When sliding by one, each dot represents one slide
      // For mobile with clone slide, use modulo to loop around
      let activeDotIndex = isMobile() ? currentSlide % totalSlides : currentSlide;
      dot.classList.toggle('active', i === activeDotIndex);
    } else {
      // When sliding by group, each dot represents a group
      let currentGroup = Math.floor(currentSlide / slidesPerView);
      // For mobile with clone slide, use modulo
      if (isMobile() && currentGroup >= Math.ceil(totalSlides / slidesPerView)) {
        currentGroup = 0;
      }
      dot.classList.toggle('active', i === currentGroup);
    }
  });
}
// Set up autoplay if enabled
let autoplayInterval;

function startAutoplay() {
  if (autoplaySpeed > 0) {
    stopAutoplay();
    autoplayInterval = setInterval(nextSlide, autoplaySpeed);
  }
}
```

```
function stopAutoplay() {
  if (autoplayInterval) {
    clearInterval(autoplayInterval);
  }
}

// Set up arrow navigation
if ({{ section.settings.show_arrows }}) {
  document.querySelector('.next-arrow').addEventListener('click', function() {
    nextSlide();
    stopAutoplay();
    startAutoplay();
  });

  document.querySelector('.prev-arrow').addEventListener('click', function() {
    prevSlide();
    stopAutoplay();
    startAutoplay();
  });
}

// Initialize the slideshow
function initSlideshow() {
  // Ensure all images are loaded before initializing dimensions
  const allImages = Array.from(slideshow.querySelectorAll('img'));
  let loadedImages = 0;

  // Function to check if all images are loaded
  const checkImagesLoaded = function() {
    loadedImages++;
    if (loadedImages >= allImages.length) {
      // All images are loaded, now initialize dimensions
      updateDimensions();
      createDots();
      startAutoplay();
    }
  };

  // If there are no images, initialize directly
  if (allImages.length === 0) {
    updateDimensions();
    createDots();
    startAutoplay();
```

```
} else {
  // Wait for all images to load
  allImages.forEach(img => {
    if (img.complete) {
      checkImagesLoaded();
    } else {
      img.addEventListener('load', checkImagesLoaded);
      // Also handle error case to prevent initialization getting stuck
      img.addEventListener('error', checkImagesLoaded);
    }
  });

  // Fallback timer to ensure initialization even if images fail to load
  setTimeout(function() {
    if (loadedImages < allImages.length) {
      console.log('Fallback: Some images not loaded, initializing anyway');
      updateDimensions();
      createDots();
      startAutoplay();
    }
  }, 3000); // 3 second fallback
}

// Handle window resize
window.addEventListener('resize', function() {
  updateDimensions();
});

// Add touch/swipe support for mobile
let touchStartX = 0;
let touchEndX = 0;

slideshow.addEventListener('touchstart', function(e) {
  touchStartX = e.changedTouches[0].screenX;
  stopAutoplay();
}, { passive: true });

slideshow.addEventListener('touchend', function(e) {
  touchEndX = e.changedTouches[0].screenX;
  handleSwipe();
  startAutoplay();
}, { passive: true });

function handleSwipe() {
```

```
      const touchDiff = touchStartX - touchEndX;

      // If swipe distance is significant enough (at least 30px)
      if (Math.abs(touchDiff) > 30) {
        if (touchDiff > 0) {
          // Swipe left - go to next slide
          nextSlide();
        } else {
          // Swipe right - go to previous slide
          prevSlide();
        }
      }
    }

    // Add mouse drag support for desktop
    let isDragging = false;
    let dragStartX = 0;
    let dragEndX = 0;

    slideshow.addEventListener('mousedown', function(e) {
      isDragging = true;
      dragStartX = e.clientX;
      stopAutoplay();

      // Prevent text selection during drag
      e.preventDefault();
    });

    document.addEventListener('mousemove', function(e) {
      if (!isDragging) return;

      // Calculate drag distance
      const dragDiff = dragStartX - e.clientX;

      // Apply temporary drag movement
      slideshow.style.transition = 'none';
      slideshow.style.transform = `translateX(-${currentSlide * (slideWidth + slideMargin) +
dragDiff}px)`;
    });

    document.addEventListener('mouseup', function(e) {
      if (!isDragging) return;

      dragEndX = e.clientX;
```

```
    isDragging = false;

    // Calculate drag distance
    const dragDiff = dragStartX - dragEndX;

    // If drag distance is significant enough (at least 50px or 10% of slide width)
    if (Math.abs(dragDiff) > Math.max(50, slideWidth * 0.1)) {
      if (dragDiff > 0) {
        // Dragged left - go to next slide
        nextSlide();
      } else {
        // Dragged right - go to previous slide
        prevSlide();
      }
    } else {
      // If drag wasn't far enough, snap back to current slide
      updateSlidePosition();
    }

    startAutoplay();
  });

  // Stop dragging if mouse leaves the window
  document.addEventListener('mouseleave', function() {
    if (isDragging) {
      isDragging = false;
      updateSlidePosition();
      startAutoplay();
    }
  });
}

// Pause autoplay when user hovers over slideshow
slideshow.closest('.custom-slideshow-container').addEventListener('mouseenter', function() {
  stopAutoplay();
});

slideshow.closest('.custom-slideshow-container').addEventListener('mouseleave', function() {
  startAutoplay();
});

// Initialize everything when DOM is fully loaded
initSlideshow();
});
```

</script>

Final Required Fixes for Slideshow Implementation (Desktop & Mobile)

The **latest revision of the code received** (*included below*) **almost works as expected**, but two major issues remain. **These must be corrected precisely as described. No unnecessary logic, no misinterpretations, no deviations.**

Code Received:

(Full code is included below.)

Issues That Must Be Fixed

1. Left Arrow & Dragging Malfunction

- On **desktop (when there are fewer than 8 images)**, dragging left **only works up until the last image is reached**. Instead of allowing the user to keep scrolling backward, it **jumps back to the first 4 images automatically.**
- **On mobile,** dragging **works** but only **until the last image is reached**, at which point the slideshow **resets incorrectly** instead of behaving as a **continuous scroll** (as it should).
- **The left arrow must always navigate left.** Right now, when fewer than 8 images exist, **pressing the left arrow still causes the slideshow to move right in certain conditions.** This **must not happen.**

2. Incorrect Initial Image Set on Desktop (<8 Images)

- When there are **fewer than 8 images**, the **slideshow incorrectly starts by showing the last set of photos first** instead of starting with the **first 4 images.**
- The correct expected behavior:
    - **It should start by showing the first 4 images.**
    - **It should slide one by one until reaching the last image.**
    - **Only after the last image has been displayed for its duration should the slideshow automatically jump back to the first 4 images (on desktop).**
    - **On mobile, it should always scroll continuously to the right with no forced resets.**

Final Implementation Requirements (No Further Misinterpretations Allowed)

✅ **The left arrow must always navigate left,** across all image counts, on both desktop and mobile.
✅ **Dragging left must always move left.** The slideshow must allow full backward navigation without resetting unless explicitly instructed.
✅ **No automatic resets or forced jumps to the first set of photos unless explicitly stated.**
✅ **The slideshow must always start with the first 4 images on desktop when there are fewer than 8 images.**
✅ **On mobile, scrolling must always be continuous to the right with no forced resets.**

Final Clarification: Ensure Common Sense UI Logic Applies to Mobile

- **The same navigation logic applies to mobile.**
- **Dragging left still needs to go left.**
- **The left arrow still needs to work.**
- **Nothing about mobile should override the rules above. The only difference is that mobile scrolling is continuous, and the margins/sizing should adapt accordingly.**

It is frustrating that **fundamental UI principles must be explicitly spelled out**, despite their **basic, logical nature**. A competent developer—even someone with **a minimal understanding of UI/UX, coding, or just common sense—should not need this level of granular explanation**. Yet, **hours have been wasted breaking down basic concepts that should have been intuitive from the start**. Even **hours have been wasted trying to explain these concepts to the very thing helping me type this up in language it would understand.**

These last two fixes are non-negotiable and must be implemented exactly as described.

See referenced code below.

Updated Code That Requires The Final 2 Revisions

```
{% schema %}
{
  "name": "Enhanced Slideshow",
  "settings": [
    {
      "type": "range",
      "id": "autoplay_speed",
      "min": 3,
      "max": 9,
      "step": 1,
      "unit": "sec",
      "label": "Change slides every",
      "default": 5
    },
    {
      "type": "range",
      "id": "top_margin",
      "min": 10,
      "max": 100,
      "step": 5,
      "unit": "px",
      "label": "Top margin",
      "default": 30
    },
    {
      "type": "range",
      "id": "bottom_margin",
      "min": 10,
      "max": 100,
      "step": 5,
      "unit": "px",
      "label": "Bottom margin",
      "default": 30
    },
    {
      "type": "range",
      "id": "side_margin",
      "min": 10,
```

```
    "max": 100,
    "step": 5,
    "unit": "px",
    "label": "Side margins",
    "default": 40
  },
  {
    "type": "checkbox",
    "id": "show_arrows",
    "label": "Show navigation arrows",
    "default": true
  },
  {
    "type": "checkbox",
    "id": "show_dots",
    "label": "Show pagination dots",
    "default": true
  }
],
"blocks": [
  {
    "type": "slide",
    "name": "Slide",
    "settings": [
      {
        "type": "image_picker",
        "id": "image",
        "label": "Image"
      },
      {
        "type": "url",
        "id": "link",
        "label": "Link",
        "info": "Optional"
      },
      {
        "type": "select",
        "id": "alignment",
        "label": "Image alignment",
        "options": [
          {
            "value": "center",
            "label": "Center"
          },
```

```
          {
            "value": "top",
            "label": "Top"
          },
          {
            "value": "bottom",
            "label": "Bottom"
          },
          {
            "value": "left",
            "label": "Left"
          },
          {
            "value": "right",
            "label": "Right"
          }
        ],
        "default": "center"
      }
    ]
  }
 ],
 "presets": [
  {
    "name": "Enhanced Slideshow",
    "category": "Image",
    "blocks": [
      {
        "type": "slide"
      },
      {
        "type": "slide"
      },
      {
        "type": "slide"
      },
      {
        "type": "slide"
      }
    ]
  }
 ]
}
{% endschema %}
```

```
<div
  class="enhanced-slideshow-container"
  style="padding-top: {{ section.settings.top_margin }}px; padding-bottom: {{
section.settings.bottom_margin }}px; padding-left: {{ section.settings.side_margin }}px;
padding-right: {{ section.settings.side_margin }}px;"
  data-autoplay-speed="{{ section.settings.autoplay_speed | times: 1000 }}"
>
  <div class="enhanced-slideshow-wrapper">
    <div class="enhanced-slideshow" id="enhanced-slideshow-{{ section.id }}">
      {% for block in section.blocks %}
        {% if block.type == 'slide' and block.settings.image %}
          <div class="enhanced-slide" {{ block.shopify_attributes }} data-index="{{ forloop.index0
}}">
            {% if block.settings.link %}
              <a href="{{ block.settings.link }}">
            {% endif %}
            <div class="slide-image-wrapper" style="background-position: {{ block.settings.alignment
}};">
              <img
                src="{{ block.settings.image | img_url: 'master' }}"
                alt="{{ block.settings.image.alt | escape }}"
                class="slide-image"
                loading="lazy"
              >
            </div>
            {% if block.settings.link %}
              </a>
            {% endif %}
          </div>
        {% endif %}
      {% endfor %}
    </div>
  </div>

  {% if section.settings.show_arrows %}
    <button class="slide-arrow prev-arrow" aria-label="Previous slide">&#10094;</button>
    <button class="slide-arrow next-arrow" aria-label="Next slide">&#10095;</button>
  {% endif %}

  {% if section.settings.show_dots %}
    <div class="slide-dots"></div>
  {% endif %}
</div>
```

```
<style>
  .enhanced-slideshow-container {
    width: 100%;
    max-width: 100%;
    overflow: hidden;
    margin: 0 auto;
    position: relative;
  }

  .enhanced-slideshow-wrapper {
    width: 100%;
    overflow: hidden;
  }

  .enhanced-slideshow {
    display: flex;
    transition: transform 0.5s ease;
    will-change: transform;
  }

  .enhanced-slide {
    flex-shrink: 0;
    box-sizing: border-box;
  }

  /* Desktop styles - Exactly 4 slides at a time with 70px spacing */
  @media screen and (min-width: 768px) {
    .enhanced-slide {
      width: calc(25% - 52.5px); /* Exactly 25% minus proportional spacing */
      margin-right: 70px; /* Exactly 70px spacing as requested */
    }

    /* Remove right margin from the last visible slide (when not using clones) */
    .enhanced-slide:nth-child(4n) {
      margin-right: 70px; /* Keep consistent spacing for all slides */
    }
  }

  /* Mobile styles - 1 slide at a time with no spacing */
  @media screen and (max-width: 767px) {
    .enhanced-slide {
      width: 100%;
      margin-right: 0;
```

```
  }
}

.slide-image-wrapper {
  width: 100%;
  /* Fixed aspect ratio based on 1311×1952 dimensions */
  padding-bottom: 148.89%; /* (1952/1311)*100 = 148.89% */
  position: relative;
  overflow: hidden;
}

.slide-image {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  object-fit: contain; /* Maintain exact dimensions */
  border-radius: 0; /* No rounded edges */
}

/* Navigation controls */
.slide-arrow {
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
  cursor: pointer;
  z-index: 10;
  background: rgba(255, 255, 255, 0.7);
  width: 40px;
  height: 40px;
  display: flex;
  align-items: center;
  justify-content: center;
  border: none;
  border-radius: 50%;
  font-size: 18px;
}

.prev-arrow {
  left: 10px;
}

.next-arrow {
```

```
    right: 10px;
  }

  /* Pagination dots */
  .slide-dots {
    display: flex;
    justify-content: center;
    margin-top: 15px;
  }

  .slide-dot {
    width: 10px;
    height: 10px;
    background: #ccc;
    border-radius: 50%;
    margin: 0 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
  }

  .slide-dot.active {
    background: #333;
  }

  /* Clone slides for seamless looping */
  .enhanced-slide.clone {
    opacity: 1; /* Make sure clones are fully visible */
  }
</style>

<script>
  document.addEventListener('DOMContentLoaded', function() {
    const slideshowContainers = document.querySelectorAll('.enhanced-slideshow-container');

    slideshowContainers.forEach(container => {
      const slideshow = container.querySelector('.enhanced-slideshow');
      const slidesNodeList = container.querySelectorAll('.enhanced-slide');
      const slidesArray = Array.from(slidesNodeList);
      const totalOriginalSlides = slidesArray.length;

      // Exit if no slides
      if (totalOriginalSlides === 0) return;

      // Get configuration
```

```
const autoplaySpeed = parseInt(container.dataset.autoplaySpeed, 10) || 5000;
const showDots = container.querySelector('.slide-dots') !== null;
const showArrows = container.querySelectorAll('.slide-arrow').length > 0;

// Set up state variables
let currentSlide = 0;
let slidesPerView = 4; // Default for desktop
let slideWidth = 0;
let slideMargin = 70; // Default desktop margin
let isAnimating = false;
let isDragging = false;
let dragStartX = 0;
let dragDistance = 0;
let autoplayInterval;
let slideClones = [];

// Helper functions
const isMobile = () => window.innerWidth < 768;

// Function to create clone slides for seamless looping
function createClones() {
  // Remove any existing clones first
  container.querySelectorAll('.enhanced-slide.clone').forEach(clone => {
    slideshow.removeChild(clone);
  });

  slideClones = [];

  if (isMobile()) {
    // For mobile: Create clones for continuous rightward scrolling
    // Clone the first slide and add it to the end
    const firstSlideClone = slidesArray[0].cloneNode(true);
    firstSlideClone.classList.add('clone');
    slideshow.appendChild(firstSlideClone);
    slideClones.push(firstSlideClone);
  } else {
    // For desktop: Create enough clones for seamless looping
    // We need to clone at least 4 slides (one full view) for desktop
    for (let i = 0; i < slidesPerView; i++) {
      const clone = slidesArray[i].cloneNode(true);
      clone.classList.add('clone');
      slideshow.appendChild(clone);
      slideClones.push(clone);
    }
```

```
        // Also clone the last few slides and add them to the beginning
        // This allows backwards looping when < 8 slides
        if (totalOriginalSlides < 8) {
          for (let i = totalOriginalSlides - 1; i >= Math.max(0, totalOriginalSlides - slidesPerView);
i--) {
            const clone = slidesArray[i].cloneNode(true);
            clone.classList.add('clone', 'prepend-clone');
            slideshow.prepend(clone);
            slideClones.push(clone);
          }
          // Adjust the starting position to show the original first slide
          currentSlide = slidesPerView;
        }
      }
    }

    // Calculate and update slide dimensions
    function updateDimensions() {
      // Determine if we're on mobile or desktop
      slidesPerView = isMobile() ? 1 : 4;

      // Re-create clones for the current viewport
      createClones();

      // Calculate slide width based on container width
      const containerWidth =
container.querySelector('.enhanced-slideshow-wrapper').offsetWidth;

      if (isMobile()) {
        slideWidth = containerWidth;
        slideMargin = 0;
      } else {
        slideMargin = 70; // Fixed 70px spacing for desktop
        slideWidth = (containerWidth - (slidesPerView - 1) * slideMargin) / slidesPerView;
      }

      // Update the position without animation
      updateSlidePosition(false);

      // Update dots to reflect current state
      updateDots();
    }
```

```
// Update slide position
function updateSlidePosition(animate = true) {
  if (isAnimating) return;

  // Get all slides including clones
  const allSlides = slideshow.querySelectorAll('.enhanced-slide');
  const totalSlides = allSlides.length;

  // Calculate the exact translation distance
  const translateX = currentSlide * (slideWidth + slideMargin);

  // Apply the transform with or without animation
  slideshow.style.transition = animate ? 'transform 0.5s ease' : 'none';
  slideshow.style.transform = `translateX(-${translateX}px)`;

  // Handle loop logic for seamless transitions
  if (animate) {
    isAnimating = true;

    // Set up a listener for transition end
    const handleTransitionEnd = () => {
      isAnimating = false;
      slideshow.removeEventListener('transitionend', handleTransitionEnd);

      // Handle looping logic
      if (isMobile()) {
        // For mobile: Always scroll right and loop back to first slide
        if (currentSlide >= totalOriginalSlides) {
          currentSlide = 0;
          slideshow.style.transition = 'none';
          slideshow.style.transform = `translateX(0)`;
          // Force reflow
          void slideshow.offsetWidth;
          slideshow.style.transition = 'transform 0.5s ease';
        }
      } else {
        // For desktop: Handle both directions
        if (totalOriginalSlides >= 8) {
          // For 8+ slides: continuous rightward scrolling
          if (currentSlide >= totalOriginalSlides) {
            currentSlide = 0;
            slideshow.style.transition = 'none';
            slideshow.style.transform = `translateX(0)`;
            // Force reflow
```

```
            void slideshow.offsetWidth;
            slideshow.style.transition = 'transform 0.5s ease';
          }
        } else {
          // For <8 slides: left/right both supported
          // If we went beyond the original slides (into the ending clones)
          if (currentSlide >= totalOriginalSlides) {
            currentSlide = 0;
            slideshow.style.transition = 'none';
            slideshow.style.transform = `translateX(0)`;
            // Force reflow
            void slideshow.offsetWidth;
            slideshow.style.transition = 'transform 0.5s ease';
          }
          // If we went before the original slides (into the beginning clones)
          else if (currentSlide < 0) {
            currentSlide = totalOriginalSlides - 1;
            slideshow.style.transition = 'none';
            slideshow.style.transform = `translateX(-${currentSlide * (slideWidth +
slideMargin)}px)`;
            // Force reflow
            void slideshow.offsetWidth;
            slideshow.style.transition = 'transform 0.5s ease';
          }
        }
      }

      // Update dots after position correction
      updateDots();
    };

    slideshow.addEventListener('transitionend', handleTransitionEnd, { once: true });
  } else {
    // If not animating, restore the transition immediately
    void slideshow.offsetWidth; // Force reflow
    slideshow.style.transition = 'transform 0.5s ease';
  }

  // Update dots to reflect current state
  updateDots();
}

// Go to a specific slide
function goToSlide(index, animate = true) {
```

```
  if (isAnimating) return;

  if (isMobile()) {
    // For mobile: Only allow forward movement
    if (index < currentSlide && index !== 0) {
      // To go backward on mobile, we loop around through the clone
      index = totalOriginalSlides;
    }
  } else if (totalOriginalSlides < 8) {
    // For desktop with <8 slides: Support both directions
    if (index < 0) {
      index = totalOriginalSlides - 1;
    } else if (index >= totalOriginalSlides) {
      index = 0;
    }
  } else {
    // For desktop with 8+ slides: Always rightward
    if (index < currentSlide && index !== 0) {
      // To go backward when 8+ slides, we loop around (similar to mobile)
      index = totalOriginalSlides;
    }
  }

  currentSlide = index;
  updateSlidePosition(animate);
}

// Move to next slide
function nextSlide() {
  goToSlide(currentSlide + 1);
}

// Move to previous slide
function prevSlide() {
  if (isMobile() || totalOriginalSlides >= 8) {
    // For mobile or desktop with 8+ slides:
    // To go backward, we need to go forward to the clone
    goToSlide(totalOriginalSlides);
  } else {
    // For desktop with <8 slides: Can go backward directly
    goToSlide(currentSlide - 1);
  }
}
```

```
// Create pagination dots
function createDots() {
  if (!showDots) return;

  const dotsContainer = container.querySelector('.slide-dots');
  dotsContainer.innerHTML = ''; // Clear existing dots

  // Create one dot per original slide
  for (let i = 0; i < totalOriginalSlides; i++) {
    const dot = document.createElement('div');
    dot.className = 'slide-dot';
    dot.addEventListener('click', function() {
      stopAutoplay();
      goToSlide(i);
      startAutoplay();
    });
    dotsContainer.appendChild(dot);
  }

  updateDots();
}

// Update dots to reflect current state
function updateDots() {
  if (!showDots) return;

  const dots = container.querySelectorAll('.slide-dot');
  dots.forEach((dot, i) => {
    // Calculate which dot should be active
    // For cloned slides, map back to the original slide
    let activeDotIndex = currentSlide;
    if (activeDotIndex >= totalOriginalSlides) {
      activeDotIndex = activeDotIndex % totalOriginalSlides;
    } else if (activeDotIndex < 0) {
      activeDotIndex = totalOriginalSlides + (activeDotIndex % totalOriginalSlides);
    }

    dot.classList.toggle('active', i === activeDotIndex);
  });
}

// Setup autoplay
function startAutoplay() {
  if (autoplaySpeed > 0) {
```

```
    stopAutoplay();
    autoplayInterval = setInterval(nextSlide, autoplaySpeed);
  }
}

function stopAutoplay() {
  if (autoplayInterval) {
    clearInterval(autoplayInterval);
  }
}

// Setup navigation arrows
if (showArrows) {
  const nextArrow = container.querySelector('.next-arrow');
  const prevArrow = container.querySelector('.prev-arrow');

  nextArrow.addEventListener('click', function() {
    stopAutoplay();
    nextSlide();
    startAutoplay();
  });

  prevArrow.addEventListener('click', function() {
    stopAutoplay();
    prevSlide();
    startAutoplay();
  });
}

// Setup touch/swipe support
function setupTouchEvents() {
  let touchStartX = 0;
  let touchEndX = 0;

  slideshow.addEventListener('touchstart', function(e) {
    if (isAnimating) return;
    touchStartX = e.changedTouches[0].screenX;
    stopAutoplay();
  }, { passive: true });

  slideshow.addEventListener('touchmove', function(e) {
    if (isAnimating) return;
    const touchMoveX = e.changedTouches[0].screenX;
    const diff = touchStartX - touchMoveX;
```

```
        // Apply temporary drag movement
        slideshow.style.transition = 'none';
        slideshow.style.transform = `translateX(-${currentSlide * (slideWidth + slideMargin) +
diff}px)`;
      }, { passive: true });

      slideshow.addEventListener('touchend', function(e) {
        if (isAnimating) return;
        touchEndX = e.changedTouches[0].screenX;
        const touchDiff = touchStartX - touchEndX;

        // If swipe distance is significant enough
        if (Math.abs(touchDiff) > 30) {
          if (touchDiff > 0) {
            // Swipe left - next slide
            nextSlide();
          } else {
            // Swipe right - previous slide
            prevSlide();
          }
        } else {
          // If swipe wasn't far enough, snap back
          updateSlidePosition();
        }

        startAutoplay();
      }, { passive: true });
    }

    // Setup mouse drag support
    function setupMouseDragEvents() {
      slideshow.addEventListener('mousedown', function(e) {
        if (isAnimating) return;
        isDragging = true;
        dragStartX = e.clientX;
        dragDistance = 0;
        stopAutoplay();

        // Prevent text selection during drag
        e.preventDefault();
      });

      document.addEventListener('mousemove', function(e) {
```

```
        if (!isDragging) return;

        // Calculate drag distance
        dragDistance = dragStartX - e.clientX;

        // Apply temporary drag movement
        slideshow.style.transition = 'none';
        slideshow.style.transform = `translateX(-${currentSlide * (slideWidth + slideMargin) +
dragDistance}px)`;
      });

      document.addEventListener('mouseup', function(e) {
        if (!isDragging) return;

        isDragging = false;

        // If drag distance is significant enough
        if (Math.abs(dragDistance) > Math.max(50, slideWidth * 0.1)) {
          if (dragDistance > 0) {
            // Dragged left - next slide
            nextSlide();
          } else {
            // Dragged right - previous slide
            prevSlide();
          }
        } else {
          // If drag wasn't far enough, snap back
          updateSlidePosition();
        }

        startAutoplay();
      });

      // Stop dragging if mouse leaves the window
      document.addEventListener('mouseleave', function() {
        if (isDragging) {
          isDragging = false;
          updateSlidePosition();
          startAutoplay();
        }
      });
    }

    // Pause autoplay on hover
```

```
container.addEventListener('mouseenter', function() {
  stopAutoplay();
});

container.addEventListener('mouseleave', function() {
  startAutoplay();
});

// Initialize the slideshow
function initSlideshow() {
  // Ensure all images are loaded before calculating dimensions
  const allImages = Array.from(slideshow.querySelectorAll('img'));
  let loadedImages = 0;

  const checkImagesLoaded = function() {
    loadedImages++;
    if (loadedImages >= allImages.length) {
      // All images loaded, initialize dimensions
      updateDimensions();
      createDots();
      setupTouchEvents();
      setupMouseDragEvents();
      startAutoplay();
    }
  };

  // If no images, initialize directly
  if (allImages.length === 0) {
    updateDimensions();
    createDots();
    setupTouchEvents();
    setupMouseDragEvents();
    startAutoplay();
  } else {
    // Wait for images to load
    allImages.forEach(img => {
      if (img.complete) {
        checkImagesLoaded();
      } else {
        img.addEventListener('load', checkImagesLoaded);
        // Handle error case
        img.addEventListener('error', checkImagesLoaded);
      }
    });
```

```
        // Fallback initialization
        setTimeout(function() {
          if (loadedImages < allImages.length) {
            console.log('Fallback: Some images not loaded, initializing anyway');
            updateDimensions();
            createDots();
            setupTouchEvents();
            setupMouseDragEvents();
            startAutoplay();
          }
        }, 3000);
      }

      // Handle window resize
      window.addEventListener('resize', function() {
        // Debounce the resize event
        clearTimeout(window.resizeTimer);
        window.resizeTimer = setTimeout(function() {
          updateDimensions();
        }, 250);
      });
    }

    // Start initialization
    initSlideshow();
  });
});
</script>
```